GitHub Uncovered: Revealing the Social Fabric of Software Development Communities

Abduljaleel Al Rubaye
University of Central Florida

GITHUB UNCOVERED:
REVEALING THE SOCIAL FABRIC OF SOFTWARE DEVELOPMENT COMMUNITIES

by

ABDULJALEEL AL RUBAYE
M.S. Florida Institute of Technology, 2016
B.S. University of Babylon, 2009

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2024

Major Professor: Gita Sukthankar

ABSTRACT

The proliferation of open-source software development platforms has given rise to various online social communities where developers can seamlessly collaborate, showcase their projects, and exchange knowledge and ideas. GitHub stands out as a preeminent platform within this ecosystem. It offers developers a space to host and disseminate their code, participate in collaborative ventures, and engage in meaningful dialogues with fellow community members.

This dissertation embarks on a comprehensive exploration of various facets of software development communities on GitHub, with a specific focus on innovation diffusion, repository popularity dynamics, code quality enhancement, and user commenting behaviors.

This dissertation introduces a popularity-based model that elucidates the diffusion of innovation on GitHub. We scrutinize the influence of a repository's popularity on the transfer of knowledge and the adoption of innovative practices, relying on a dataset encompassing GitHub fork events. Through a meticulous analysis of developers' collaborative coding efforts, this dissertation furnishes valuable insights into the impact of social factors, particularly popularity, on the diffusion of innovation.

Furthermore, we introduce a novel approach to computing a weight-based popularity score, denoted as the Weighted Trend Popularity Score (WTPS), derived from the historical trajectory of repository popularity indicators, such as fork and star counts. The accuracy of WTPS as a comprehensive repository popularity indicator is assessed, and the significance of having a singular metric to represent repository popularity is underscored.

We delve into the realm of code quality on GitHub by examining it from the perspective of code reviews. Our analysis centers on understanding the code review process and presents an approach rooted in regularity to foster superior code quality by enforcing coding standards.

In the concluding phase of our research, we investigate the intricacies of communication within technology-related online communities. Our attention is drawn to the impact of user popularity on communication, as elucidated through an examination of comment timelines and commenting communities. To contextualize our findings, we compare the behavioral patterns of GitHub developers and users on other platforms, such as Reddit and Stack Overflow.

To the Master of our time, Imam Al Mahdi (Peace Be Upon Him)...

ACKNOWLEDGMENTS

I extend my deepest gratitude to my father, Shaikh Jameel Al Rubaye, and my mother, Al Hajjah Um Ali, for their unwavering support and encouragement. I extend a special thank you to my beloved mother, who passed away recently, just before the final stage of my PhD journey. Her encouragement was the main driving force behind my perseverance. Though she is no longer with us, her belief in me continues to inspire and guide me. I am forever grateful for her love and sacrifice.

I also want to express my heartfelt gratitude to my family, especially my wife, Hoda, and children, Sadeq, Fatima, and Ali, for their great support during this challenging time. Their encouragement and love have been a significant source of motivation.

I am profoundly thankful to my advisor, Dr. Gita Sukthankar, for her guidance, encouragement, and mentorship. I learned a lot from Dr. Sukthankar, and I am so thankful that I could work and finish my research under her supervision. I also want to thank my committee members for their feedback, criticism, and guidance, and the Department of Computer Science at the University of Central Florida for their support in helping me achieve my academic goal.

Finally, I want to thank my extended family, friends, and all who have contributed. Your encouragement has been instrumental, and I am truly grateful for your role in my success.

TABLE OF CONTENTS

LIST OF FIGURES xiii

LIST OF TABLES xix

CHAPTER 1: INTRODUCTION 1

Social Groups 1

Online Social Networks 2

Social Networks and Developers 3

GitHub: A Social Platform for Developers 4

Problem Statement 7

Research Questions 8

CHAPTER 2: LITERATURE REVIEW 10

Diffusion of Innovation 10

Popularity 13

Collaboration and Communication 15

Social Roles 19

Code Review and Code Quality 22

Visualization 24

CHAPTER 3: A POPULARITY-BASED MODEL OF THE DIFFUSION OF INNOVATION ON GITHUB 27

Introduction 27

Method 28

Data Filtration 29

Model Construction 30

Identifying Structural Changes over Time 32

Results 36

Innovation Adoption 37

Sociality 38

Conclusion 39

Algorithms 43

CHAPTER 4: POPULARITY SCORE IN GITHUB 45

Introduction 45

Method 46

Dataset 46

Popularity Growth 48

Calculating Popularity Score 50

Popularity Weight 51

Weighted Total Popularity Score ($WTPS$) 53

Results 54

Ranking Popularity 54

$WTPS$ vs. other popularity indicators 56

Popularity in Collaboration Networks 58

Conclusion 61

CHAPTER 5: IMPROVING CODE QUALITY IN GITHUB 64

Introduction 64

Method: 67

Dataset 67

Code Review Process via Issue Tracking mechanism 74

Collaborators and Reviewers 76

Issues Density 77

Issues Opening Regularity 80

Results 81

Issue Regularity 81

Issues Community 86

Expertise Coverage 90

Developers Popularity 92

Threats to Validity 94

Conclusion 94

Algorithm 96

CHAPTER 6: CONVERSATIONS AND INTERCONNECTIONS IN TECH-RELATED ONLINE COMMUNITIES 97

Introduction 97

Method: 99

Dataset 99

Comment Timeline 102

Users Popularity 104

Community Detection 105

Social Roles Categorization 105
Results 108
Commenting Behaviors 108
Users Popularity Effect 110
Authors' Popularity Effect 111
MPCs' Popularity Effect 112
Social Commenting Communities 117
Commenters Interconnection 120
Commenting Community Effective Engagement Score (CCEE) 122
Conclusion 126
Threats To Validity 128
CHAPTER 7: CONCLUSION 129
Main Contributions 129
Modeling the Diffusion of Innovation in GitHub 129
The popularity of GitHub repositories 130
Code Quality in GitHub 131
Commenting Behaviors in Social Communities of Developers 132

Potential Limitations 134

Future Works 134

Potential Applications 135

LIST OF REFERENCES 138

LIST OF FIGURES

Figure 3.1: Our model contains three components (repos, actors, and followers) and three types of connections: (a) repo-repo (b) repo-actor, and (c) actor-follower. 31

Figure 3.2: Repo-follower interaction: a probable connection that may occur when follower f of actor A forks repository R. 32

Figure 3.3: The correlation between the number of fork events and the watcher count of the repositories in our dataset 34

Figure 3.4: The popularity-based network evolution process is generated with the data in Table 3.2. (t_0) is the base model generated according to the data shown in Table 3.2; in (t_1) R_1 is the most popular repository, so the likelihood of it being forked by f_1 is higher. In (t_2) R_3 will have a higher chance of being forked and in (t_3) R_2. Notice that the temporal distance between the time points is the time required for one repository to be forked. 35

Figure 3.5: Shows the initial network of connections extracted from our GitHub fork event dataset. The colors indicate different communities in the network, which were detected with the modularity algorithm [1]. Red indicates higher connectivity among the nodes and blue marks less connected communities. The node size indicates the nodes' type (e.g., repository nodes' size > size of actor nodes > size of followers nodes). We used the open source software Gephi [2] to visualize this network (V=$19,250, E = 33,988$) 36

Figure 3.6: The repositories' rate of acquiring new fork events in the popularity and random models at three-time points, where the repository popularity is assumed to be the total number of fork events that occurred since creation. (a) depicts the difference between popularity and random structures at t_1 against the structure at t_0, while (b) shows the same comparison for the structures at t_2, as well as the comparison in (c) at t_3. 38

Figure 3.7: Power-law fitting of both popularity (prob) and random models degree distributions at all time points. 40

Figure 4.1: Distribution of the repositories features: forks, stars, watchers, age of repositories, size, and the number of repository owners' followers. 47

Figure 4.2: Presents a sample JSON object that represents the needed information for each repository, which was extracted from the collected dataset. 47

Figure 4.3: Correlation of age and popularity measurements. (a) shows age-stars correlation with a coefficient of 0.3215. (b) shows the age-forks correlation with a coefficient of 0.2923, and (c) shows the age-watchers correlation with a coefficient of 0.3162 [3] 49

Figure 4.4: Shows the *stargazer's* growth of a randomly picked GitHub repository. 50

Figure 4.5: The popularity score is extended from the combination of some of the main popularity indicators of a GitHub repository. 51

Figure 4.6: The rank of repositories of Table 4.1 based on their popularity indicator forks, stars, and $WTPS$. 54

Figure 4.7: A sample of 20 repositories' popularity ranking using different popularity measures: forks, stars, and $WTPS$ (t= one month). 55

Figure 4.8: WTPS-Forks and $WTPS$-Stars correlation. 56

Figure 4.9: Subset of an example network: repository, owners' followers, and the repository-follower connections. 59

Figure 4.10A sample graph of 5000 nodes and 5195 edges, where the red nodes represent the repositories and the blue ones are the repository owners' followers. The size of the repository nodes also is related to their popularity. This graph used $WTPS$ as a measurement of the popularity of the repository. We used the open-source software Gephi [2] to visualize this network. 60

Figure 4.11Repository deletion impact on one of the principal network characteristics: clustering coefficient. 61

Figure 5.1: Shows the mechanism of Issue Tracker on GitHub. 66

Figure 5.2: Shows a sub-diagram of (Repository-Issue) entities' relation in GitHub. Some of the field names have been modified for simplicity. 67

Figure 5.3: Presents a sample JSON object representing the needed information for each repository extracted from the collected dataset. 69

Figure 5.4: Depicts the distribution of the following attributes across all three types of repositories: Age, Opened Issues, Stargazers count, and Contributors count. . 71

Figure 5.5: Top row: Correlation between opened issues and total contributors. Bottom row: Correlation between opened issues and total repository stars. 73

Figure 5.6: Presents the correlation between the number of repository stargazers and the total number of comments received on Issue. 75

Figure 5.7: A repository Issues openings events. $a = \{I_1, I_2, I_3, I_4\}$ and $b = \{\}$ while $d_a < d_b$. 78

Figure 5.8: Displays a left-skewed distribution of the time intervals between open Issues. 79

Figure 5.9: Shows the New Issues Notifier (NIN) mechanism. 82

Figure 5.10Presents the New Issue Notifier mechanism (NIN) over a GitHub repository timeline. The *IDM* stands for Issue Distances Median and recalculates each time a new Issue is opened. 82

Figure 5.11Shows the percentage of the Issues that fall within the regular area $[median - \alpha, median + \alpha]$, where α = 6 hours, for all three types of repositories before and after running the simulator (for three different acceptance probabilities). 85

Figure 5.12A sample Issue Community (*IC*) network is formed by two types of nodes that represent Issues and reviewers. There are two layers: Layer 1 shows the edges of type e_1 that connect Reviewers to Issues, and Layer 2 shows the edges of type e_2 that connect Issue to Issue. 87

Figure 5.13Displays sub-graphs of three different Issue Communities, with just the e_1 edges shown (reviewers-Issues connections). The red nodes represent the reviewers, while the blue nodes represent the open Issues. The degree of a node is determined by its size. 88

Figure 5.14Shows the Issues coverage by Reviewers for all three types of repositories. The reviewers are sorted from high to low by popularity (follower count). . . 91

Figure 5.15Illustrates the correlation between Issues openers' popularity and the comments they receive, sorted by Issue count. The popularity of a user is its total number of followers on GitHub. 93

Figure 6.1: The number of retrieved *Posts* from GitHub, Reddit, and Stack Overflow grouped by the search keywords: "deep learning," "ROS," "robotics," "reinforcement learning," "machine learning," and "artificial intelligence." 100

Figure 6.2: The figure shows the comment timeline of a randomly picked GitHub Issue, where it illustrates the comments counts per day throughout the Issue's lifetime.103

Figure 6.3: A sample plot of the commenters of a post, classified into four different social role classes based on the users' popularity and their account age, using the k-means Classifier. 106

Figure 6.4: The cumulative commenters distribution based on their social role categories. 107

Figure 6.5: The distribution of the retrieved posts across all three platforms based on the total number of the extracted comments. 108

Figure 6.6: Illustrates distributions of the captured posts across all three platforms based on the ratio of the one-time commenters. 109

Figure 6.7: Depicts the log-log plot of authors' popularity correlation with the average commenters' popularity across the platforms. 112

Figure 6.8: Illustrates 3 sample comment timelines, each showing one of the comment tail length categories: Long, mid, and short. The comment tail includes all comments that are posted after the MPC comments. 114

Figure 6.9: Shows users' popularity in three categories across the platforms: Authors, Commenters, and MPCs (the Most Popular Commenters). 116

Figure 6.10Shows three conditions to define links in the commenting network. The nodes can have a direct weighted link through social tagging, conversation involvement, and directly responding to comments. 118

Figure 6.11Depicts the constructed networks' node-specific measurements: weighted out-degree and weighted in-degree by social roles based on the user community detection methods: Louvain, Statistical Inference, and Label Propagation. 121

Figure 6.12Illustrates a GitHub commenting community (detected using Louvain method) where nodes are sized based on three measurements: (a) the size of the nodes is proportional to the betweenness value, (b) the nodes' sizes indicating the value of the clustering coefficient, and (c) shows the nodes where they are sized based on CCEE score. 124

Figure 6.13Illustrates one of the detected commenters communities in GitHub, which was uncovered using the label propagation method. The node size is relative to the CCEE score of the users in this sub-community. 125

Figure 6.14Shows the average commenting community effective engagement score (CCEE) categorized by social roles across all platforms' detected communities. 125

LIST OF TABLES

Table 3.1: Top 10 repository languages, ranked by number of fork events 30

Table 3.2: A set of fork events is used to evolve the sample network (t_0) shown in Figure 3.4. The table presents a set of 3 events done by the actors A_1 and A_2 to fork the repositories R_1 and R_2. The table also shows two extra details: the followers of the actors and the fork event count for each repository. 35

Table 3.3: Characteristics of the evolving networks at different timesteps. 39

Table 4.1: A sample dataset of four repositories and their collected forks and stars at 5-time points [t_1... t_5] and their calculated weight W_F and W_S using Equations 4.1 and 4.2 52

Table 4.2: Calculating the Weighted Total Popularity Score ($WTPS$) for the repositories detailed in Table 4.1 based on Equation 4.4 53

Table 4.3: Correlations of $WTPS$ and repository properties 57

Table 4.4: The calculation of a regression line for popularity indicators forks, stars, and watchers and $WTPS$ based on four different time intervals. 58

Table 5.1: Top 10 topics labeled on ROS-related repositories. 70

Table 5.2: General statistics of the collected repositories. 72

Table 5.3: The Correlation between the total Issues counts to other features for all three different types of the collected repositories. 73

Table 5.4: The followers count for users open/close Issues and Issue reviewers. 77

Table 5.5: The distribution of Issues opening time. 79

Table 5.6: The percentage of dense, regular, and dispersed Issues after running the simulator for each acceptance probability (AP). 84

Table 5.7: The Correlation of repositories properties: Issues, Contributors, Stars, Issue Comments and the Issue community score. 89

Table 5.8: Shows the popularity ratio and percentage of the Issues that 20% of most popular reviewers cover. On average, 20% of the reviewers (the most popular ones) cover more than half of the Issues. 91

Table 6.1: Comment-related statistics extracted from the captured datasets. 101

Table 6.2: The average ratio of the social roles that is categorized based on users' popularity and user account age. 107

Table 6.3: Author's popularity correlations to other comment-related measurements. . . 111

Table 6.4: MPC Comment tail statistics 115

Table 6.5: MPC popularity correlations to other users' communications features. 116

Table 6.6: The detected community sizes (number of the nodes) by using the community detection method. 120

CHAPTER 1: INTRODUCTION

Social Groups

Social groups are a fundamental aspect of human society, defined as individuals interacting with each other and sharing common characteristics and values. According to a study by sociologist George Homans, social groups comprise four key elements: social interaction, shared goals, standard norms, and a sense of belonging [4]. Another theory by sociologist Ferdinand Tönnies further characterized social groups by strong emotional bonds and a sense of community, or by more impersonal and instrumental relationships [5]. Social groups can vary significantly in size and structure, from small, informal groups such as a close-knit group of friends or players of a soccer team to large, formal organizations like corporations or political parties.

In recent years, social groups have become widely used and exciting fields for research and innovation. In order to understand how much social groups impact individuals in the community, it is essential to understand their nature, structure, and dynamics. Researching social groups, such as online communities, local neighborhoods, and organizations, helps to recognize patterns of community interactions. This may lead to an understanding of the factors that affect behaviors in social communities. Knowing such factors can give insights into how social communities grow and how external elements impact them. Understanding social community structure helps scientists design and develop more effective social systems like online platforms and social media networks.

Academics from various fields, including psychology, sociology, marketing, and computer science, are attracted to understanding user behaviors in society. User behavior in society is an essential subject for research, with implications for many subjects and the ability to significantly influence society and the environment in which we all live [6]. People's actions, choices, and interactions can reveal social norms, values, and attitudes and offer valuable information about the factors influencing individual behavior [7].

Researchers can now investigate how individuals interact with one another and use technology in various contexts, including online communities and social networks, which gives them a better knowledge of these communities' power relationships and social norms and how they may affect user behavior [8].

Online Social Networks

The internet has proliferated since the late 20th century due to the widespread use of personal computers and technological advancements. This transformation has changed how people communicate, access information, and run their businesses. Internet users worldwide have increased from 738 million in 2000 to over 4.9 billion in 2020. The internet's growth has also profoundly impacted the global economy with the rise of e-commerce and the digitization of various industries.

Online social networks have become an indispensable part of daily life for most people, playing a vital role in creating virtual communities. More than 4.26 billion individuals used social media in 2021, expected to rise to approximately six billion by 2027 [9]. By providing the ability to create profiles, share multimedia content, and connect with others who share similar interests, online social networks have changed how people interact and communicate. These networks also help promote social well-being by developing meaningful relationships, which can help build and strengthen their sense of belonging to a community. Members of these communities can share knowledge and data, collaborate, and foster empathy and amicability among participants, all while building interpersonal connections and relationships [10].

Numerous online social networks attracted the attention and the interest of millions of users worldwide, including Facebook [11], Twitter [12], Instagram [13], and LinkedIn [14], each with their unique set of features, functionalities, and interfaces, offering different ways of communication and engagement in online social activities.

Social Networks and Developers

Online social networks are an essential part of the professional career of software developers and engineers. Software developers can utilize social platforms, such as LinkedIn and GitHub, to participate in online communities, join projects, showcase their work, collaborate, acquire knowledge, and share their ideas. There are other online networks like Stack Overflow [15], Reddit [16], and Quora [17] that provide platforms for individuals with different backgrounds and expertise to learn, exchange knowledge, and discuss various technology topics.

Communities centered around a shared interest in technology can create a sense of belonging and offer a supportive environment. Social platforms designed for software developers have intricate social structures, ranging from tightly connected communities to large, open networks, that can significantly influence community dynamics, such as the degree of collaboration, information sharing, and knowledge exchange. Moreover, group structures can impact how connected and involved members feel with the community, with groups centered around specific projects or initiatives fostering collaboration and teamwork. In contrast, groups focused on discussion create a more open environment focusing on sharing and learning.

The study of online social groups of developers, as online social platforms have become increasingly popular for developers to connect, collaborate, and exchange knowledge. It is essential to understand the functions and dynamics of these groups to gain insights into the software development industry and the role that online communities play in shaping it. Performing more studies and research on online social groups of developers may also provide the researchers with more valuable information on their motivations and behaviors [18].

In addition, by studying the networks of software developers, the researchers can explore how these communities form and connect, where developers can share ideas and work together on innovative projects, that can significantly impact the future of technology. Moreover, the motivations and behaviors of developers in these online communities are valuable information resources for data scientists. This information may be utilized to create developer communities and support networks that are more productive, fostering innovation, knowledge sharing, and improved cooperation. Furthermore, data scientists are also interested in exploring the impact of online social groups on the software development industry. Through their research, data scientists can gain insights into how online communities are shaping the future of technology and what impact these communities have on the software development process [19].

GitHub: A Social Platform for Developers

In this work, our main study is focused on GitHub. The GitHub platform has become a top-rated platform used by millions of developers worldwide, where they can host their code-base and showcase their abilities and skill set in program development and problem-solving. However, GitHub's socializing feature is another essential factor that made it a place for developers to interact.

GitHub is a social community of developers who can collaborate on software projects, share their code, and exchange methods for solving coding issues. They also can construct networks of expertise with other developers where they can join others to work together and contribute to software development. The social aspect of GitHub lets them share their thoughts and best methods, spread knowledge, and learn from each other. These interactions also generate essential data that can be valuable resources for researchers who want to study various aspects of software development, including collaboration patterns, code quality, and the diffusion of innovation. This data can provide valuable insights for software development teams and inform the design of new tools and processes [20].

GitHub provides various events that allow developers to track changes, collaborate, and communicate effectively. Events in GitHub refer to actions that occur within a repository. Some of the most common types of events in GitHub are:

- *Push events*: are triggered when a user pushes code to a repository and can be used to track the code changes, review code, and collaborate with other users.
- *Pull request events*: are triggered when a user submits a pull request to merge code changes into a repository where they can be used to review code, provide feedback, and collaborate with other users on the developed code.
- *Issue events*: are triggered when a user opens an issue in a repository and are used to track bugs, feature requests, and other issues related to the code being developed.
- *Comment events*: are triggered when a user adds a comment to an issue, submits a pull request, or commits in a repository and are used to provide feedback, discuss issues, and collaborate with other users.
- *Fork events*: are triggered when a user creates a copy of a repository where they can be utilized to track forks, collaborate with other users, and contribute to project development.
- *Watch events*: are triggered when a user starts watching a repository where they can be used to track activity in a repository, receive notifications, and collaborate with other users.

- *Release events*: are triggered when a user creates a new project release. They can also track releases, collaborate with other users, and distribute software updates.

Through the GitHub API, events that are recorded and made available to users play an essential role in software development, as they enable developers to track changes, collaborate more effectively, and manage events using tools like webhooks and workflows, which allow users to automate tasks and integrate with other tools, ultimately enabling them to work together, track progress, and respond to issues promptly. The collaborative software development process heavily relies on the relationship between GitHub events and user communications.

Users on GitHub can communicate with each other through events, which generate notifications visible to other users in the repository. These events are triggered whenever a user opens an issue, adds a comment, or submits a pull request. As a result, users can collaborate, provide feedback, and communicate with one another about the code being developed. GitHub also offers email notifications, push notifications, and webhooks to manage these event-related notifications, providing users with efficient communication tools and real-time updates. On the other hand, developers can also effectively manage their projects using a wide set of features provided by GitHub, such as branching and merging.

Similar platforms, like Stack Overflow and Reddit, also have a large user base of technologists. Stack Overflow is a question-and-answer website where developers can ask and answer technical questions related to programming [21]. Reddit is a social news aggregation and discussion platform where users can post links and comments on many topics, including technology [22].

Looking further into the future, GitHub social groups will continue to be important in the software development process. With attracting a significant number of users and the increasing complexity of software development, social collaboration will become increasingly critical in the future [23].

Researchers will continue to investigate the social aspects of GitHub and other similar platforms to gain insights into how social communities can be fostered and utilized for effective knowledge transfer and collaboration.

Problem Statement

The primary purpose of this dissertation is to explore the online communities on GitHub as a software development network and provide analytical work on the social nature of these communities. To achieve this general objective, we offer the following contributions:

- This study focuses on revealing the effect of GitHub's social system on code adoption by using a popularity-based approach to model the spread of innovation. We propose a novel popularity-based methodology to forecast code adoption, assessed by the frequency of fork events. Our model leverages the interconnections between GitHub users, repositories, and followers to build event timelines and social structures. We investigate multiple social factors that affect code adoption, such as the repository owner's characteristics, collaborators, and metrics like the number of stars, watchers, and forks.
- Additionally, our research focuses on the dynamics of popularity and social engagement on the GitHub platform to gain insights into how repository popularity evolves over time. We explore the influence of critical metrics such as forks, stars, and watchers on repository popularity. To accurately gauge the popularity of a repository, we introduce a novel weight-based score that considers the repository's popularity history. This score is a more reliable indicator of popularity, especially when assessing how a repository's popularity changes over time. Our study provides a new approach to evaluating and understanding the popularity of GitHub repositories.

- Furthermore, the study examines how code review can be improved by analyzing the relationship between Issue frequency and community interaction, the involvement of experienced reviewers in Issues, and the correlation between the level of experience and the number of comments received by GitHub users. We present a novel approach for improving the code review process by promoting regularity and community involvement. Our new proposed mechanism nudges developers towards greater regularity in the review process by prompting them to open issues during specific periods based on previous issue-opening behaviors by the developers. To assess the new approach, we also introduce a new metric that indicates users' involvement to evaluate the community's involvement in the code review process.
- Finally, the dissertation aims to explore social networks' effects on software development, identify the social roles and online communities in software development on GitHub, and compare it to communities of users on two other social platforms: Reddit and Stack Overflow. By analyzing users' commenting behavior, we aim to comprehend how individuals interact and evaluate the level of engagement within online social groups of developers. We also examine how user popularity affects commenting behavior. We define social roles based on user characteristics to better understand users' commenting behavior on online social networks. We also introduce a new metric to evaluate the effectiveness of user engagement within these communities. This score identifies individuals who are highly engaged in communication with their teammates and well-centered within their communities.

Research Questions

The dissertation seeks to provide insights into the dynamics of software development communities on GitHub to study the diffusion of innovation, popularity measurements, the code review process, and commenting behaviors. This study has important implications for developers, software project managers, and technical decision-makers in development communities. In this study, the main aim is to answer the following questions:

- How does the social communication of software developers influence the adoption of code and knowledge spread in GitHub?
- What are the characteristics of popular GitHub repositories, and how can we accurately measure their popularity over time?
- How can the code review process and community interaction be improved on GitHub based on Issue-related team behaviors and the popularity of the reviewers?
- How does users' popularity impact their commenting behavior and involvement in tech-related social communities?

Each question will be addressed in a separate chapter. To fulfill the objective of answering the last two questions, we will further elaborate on each question by dividing them into more specific research questions in their corresponding chapters.

CHAPTER 2: LITERATURE REVIEW

The growth in popularity of online social networks in recent years has made them an integral part of modern-day communication and information sharing, as these platforms have provided a means for individuals to connect and share information on a global scale, and concurrently, the number of studies examining various elements of these platforms has also increased. These networks provide substantial, extensive data resources that can be easily extracted and applied to important projects such as health-related [24], political [25], and economic [26] studies.

In recent years, GitHub has become a top-tier online platform, providing a valuable environment for researchers to study the social networks of developers. The vast amounts of data generated by the developers on this platform have led to an increasing interest in researching GitHub as a social network where these data can give insights into users' social behavior, communication patterns, and the factors that contribute to the popularity of repositories.

This literature review provides an overview of the existing research in these areas. It will be divided into several sections related to knowledge transfer and diffusion of innovation, the factors that contribute to the popularity of online social networks, collaborations and communication patterns, social role exploration, social contributions through the code review process, and code quality, in addition to visualization techniques and methods.

Diffusion of Innovation

Online social networks such as GitHub have become an essential platform for developers to share and disseminate knowledge. The speed and ease of knowledge spread in online social networks have facilitated the sharing of ideas, experiences, and best practices among developers, making it an attractive environment for researchers to study knowledge transfer. This section will explore the literature on the diffusion of innovation and knowledge spread in online social networks.

Yu et al. (2014) studied how social communication and innovation diffusion occur on GitHub by analyzing the growth curves and follower networks. To identify differences, they compared the growth modes of projects and users on GitHub with those of three traditional open-source software communities. The authors found that user growth on GitHub is exceptionally rapid, which they explained through the diffusion of innovation theory. Moreover, they constructed follow-networks based on the following behaviors among developers on GitHub. From this analysis, by mining follow-networks, the authors presented different social behaviors, including patterns related to users' staring behavior [27].

In their study, Zhang and colleagues (2012) examined the nature of open-source communities as dynamic and heterogeneous networks of informal knowledge exchange. They argued that the generative diffusion of innovation in such communities is influenced by the interactions within multiple knowledge networks where abundant artifacts, such as open-source software, are embedded. Specifically, they identified two significant types of networks that affect innovation diffusion: developer-developer and developer-project. These networks differ in terms of the types of knowledge transfers they emphasize and their effects on the generative diffusion of innovation. Their findings suggest that understanding the interplay between these networks and the abundant artifacts can provide insights into how innovation diffusion can be effectively facilitated within open-source communities. The implications of their research extend beyond the realm of software development and can be applied to other domains where informal knowledge networks play a crucial role in innovation diffusion [28].

Akula et al. (2019) presented a machine-learning model that predicts the spread of information within the GitHub platform. The authors focused on predicting the number of forks a given repository will receive in the future, an essential indicator of information diffusion on the platform. The authors developed a model that utilizes a variety of features, including repository characteristics, user characteristics, and social network features, to predict the number of forks a repository will receive. The model is trained on a large dataset of repositories and validated through cross-validation techniques. The authors argued that the ability to predict information diffusion on GitHub has essential implications for understanding how information spreads within software development communities. Additionally, the authors suggested extending their approach to other online platforms to understand information diffusion and social influence in various contexts [29].

In their paper published in the Journal of Systems and Software, Tantisuwankul et al. (2019) conducted a topological analysis of communication channels for knowledge sharing in recent GitHub projects. The study aimed to identify the communication patterns and structures in GitHub projects and explore how these patterns affect knowledge sharing among project members. The authors analyzed 39,906 communication events from 87 GitHub projects and applied social network and topological data analysis to the resulting data. The findings showed that communication patterns in GitHub projects are highly diverse and can take many forms, including centralized and decentralized structures. The study also revealed that project members who act as knowledge brokers, bridging gaps between different sub-groups within a project, play a significant role in promoting knowledge sharing. The authors concluded that the topological analysis of communication channels provides valuable insights into the dynamics of knowledge sharing in collaborative software development projects [30].

Acemoglu et al. (2011) studied the diffusion of innovation in social networks. The authors explored the spread of innovations across social networks by modeling the interaction between the network's individuals and the innovation's adoption process. They proposed a mathematical model that incorporated network structure, individual preferences, and adoption mechanisms and analyzed the impact of these factors on the diffusion process. The study demonstrated that the structure of the social network plays a crucial role in the diffusion of innovations, and specific network structures can facilitate or impede the adoption process. The authors also highlighted the importance of considering individual preferences and the adoption mechanism in understanding the diffusion of innovation in social networks [31].

Faraj et al. (2011) examined knowledge collaboration in online communities by investigating how knowledge is shared online and how online communities' social context can influence knowledge collaboration. The authors conducted a qualitative study of an online community of software developers. They found that knowledge collaboration in online communities is characterized by emergent, self-organized patterns of interaction that are only sometimes aligned with the community's goals. They also found that the social context of online communities, including social norms and the quality of interpersonal relationships, plays a critical role in facilitating or hindering knowledge collaboration [32].

Popularity

Popularity has become a critical factor in online social networks. Investigating the popularity of online social networks like GitHub is important for researchers as it provides an environment to examine social and economic mechanisms that drive collaboration and innovation in the digital age.

Borges et al. (2015) present a framework for evaluating the popularity of GitHub software systems based on the number of stars they receive. Additionally, they propose a taxonomy of popularity growth patterns that describe the temporal evolution of a system's star count. Through empirical analysis, the authors find a significant correlation between stars and other metrics, such as forks and third-party software usage. These results demonstrate the importance of stars as a measure of software popularity and the potential impact of third-party usage on system growth. The authors apply their framework to real-world GitHub data to evaluate and illustrate its effectiveness [33].

In another work, Borges et al. (2016) employed a statistical method based on multiple linear regression to predict the future popularity of repositories on GitHub. Their analysis revealed that the proposed models were highly accurate in predicting the number of stars a repository would receive in the future after being trained on the repository's star count over the previous six months. Furthermore, the researchers discovered that repositories with a lower growth rate or fewer stars could benefit from using specialized models generated from data collected from repositories with similar characteristics, improving the precision of the predictions. They also evaluated the model's ability to forecast a repository's rank among all the repositories on GitHub rather than just its star count. Their analysis showed a strong correlation between predicted and actual rankings, indicating that the model could be helpful in identifying repositories with the potential to become highly popular [34].

Borges et al. (2016) also conducted another study on the popularity of software systems hosted on GitHub. The study aimed to identify the main factors that impact the popularity of GitHub repositories and to understand the growth patterns of popular repositories over time. The authors investigated the growth of repositories over time and identified the main pattern of popularity as the number of repository stargazers. Additionally, the authors explored the impact of programming language and application domain on project popularity. They found that these factors significantly impacted the number of stars a GitHub project received. Furthermore, the authors studied the impact of new features on project popularity and identified four main patterns of popularity growth. These patterns were derived by clustering the time series data of over 2000 popular GitHub repositories [35].

Xavier et al. (2014) conducted an exploratory study on the factors influencing developers' popularity on GitHub, a prominent social network for software developers. The study aimed to provide valuable insights that could help individuals optimize their online presence and enhance their popularity. Using user-level features, the authors measured popularity by analyzing the number of followers and focused on a subset of the GitHub database to better understand the high popularity phenomenon. The authors discovered that commit activity is a significant predictor of popularity, with a strong correlation between repository popularity and the number of commits made by developers. However, the authors also identified cases where low-activity developers had high popularity.

Their findings underscore the need for a holistic approach to understanding the various factors contributing to GitHub's popularity. They argued that by adopting such an approach, developers could gain a deeper understanding of the platform's dynamics and develop data-driven strategies that help them enhance their online visibility and popularity [36].

In addition to these studies, several online projects and applications rank repositories by different popularity measurements. For instance, GitHub ranks the rock-star repositories in descending order by their daily stars only [37]. *Git Awards* [38] is a web-based application that gives a repository's ranking on GitHub by language or geolocation based on the number of stars. *Git Most Wanted* [39] is another web-based application that ranks the repositories based on their stars or forks counts.

Collaboration and Communication

Another exciting research subject is studying users' communication and collaborations on online social networks. Understanding the factors that contribute to successful collaborations, meaningful contributions, and productive conversations can provide insights into the dynamics of the development community and help identify potential areas for improvement.

In their work, Pace et al. [40] studied the conversations on online social networks from two perspectives: dialogic and dialectic. They analyzed the conversation structures to study the preferences of online users for participating in conversations. Their study's result showed that social network users have a greater tendency to converse with each other instead of dialectic conversations, which encourages vertical communications with users from outside their community.

Since GitHub is a rich source of data on the performance of virtual teams, many studies have centered on collaboration between developers. Dabbish et al. (2012) conducted a study investigating the role of social transparency in promoting collaborations on the GitHub platform. The authors hypothesized that the social transparency of the platform was vital for promoting collaborations by empowering users to make inferences about other developers' commitment and work quality. Through the interviews, the researchers found that users of the GitHub platform make a surprisingly rich set of social inferences from the networked activity information available on the site. The study identified four features of visual feedback that drove a rich set of inferences around commitment, work quality, community significance, and personal relevance. These inferences supported collaboration, learning, and reputation management in the community. The authors concluded that their findings could inform the design of social media platforms for large-scale collaboration and suggest various ways transparency could support innovation, knowledge sharing, and community building [23].

Saadat et al. (2018) proposed a method to improve agent-based simulations using large datasets. They initialized the simulation with archetypes representing different types of users and developed a model to simulate developer activities. They used archetypes extracted from stable clusters of GitHub activity profiles to represent the population. The researchers showed that clustering could be used to discover these archetypes and evaluated the benefits of selecting stable clusters over time. They found that simulation runs with clustering archetypes were more effective at predicting large-scale activity patterns. In particular, stable cluster-based user archetypes outperformed less stable clusters in predicting the dispersion of activity across users. While these archetypes improved the dispersion across repositories, the heuristics for assigning users to repositories did not perform as well. Overall, the study demonstrated the usefulness of using clustering to identify archetypes and the importance of stability in selecting these archetypes for initializing agent-based simulations [41].

Destefanis et al. (2018) introduce a framework for measuring the emotional effect of commenters on GitHub issues based on the theory of affective computing. They applied this framework to a dataset of GitHub issues and comments, constructing collaboration networks of users and commenters to investigate the contribution of commenters to the project and their emotional effectiveness. The authors calculated and compared the sentiment, politeness, and emotions expressed in comments written by users and commenters, using several metrics such as valence, arousal, and dominance. Their analysis revealed that commenters expressed various emotions, including anger, frustration, and gratitude. The authors emphasize the importance of understanding the emotional effects of commenters. They suggest that software developers and project managers should be aware of the emotional tone of comments and take measures to promote positive emotions and mitigate negative ones. They also propose developing automated tools to recognize emotional effects in comments and provide real-time feedback to commenters to improve communication and collaboration among team members [42].

Much research has been done on users' communication and commenting behavior on various online platforms, including Reddit and Stack Overflow. These studies have provided valuable insights into how users interact and the factors contributing to successful communication and collaboration on these platforms.

In recent years, Stack Overflow has become a popular platform for software developers to seek help and exchange knowledge. However, while Stack Overflow answers can be helpful, the study by Zhang et al. (2019) shows that relying solely on them can lead to incomplete or incorrect solutions. The authors studied and analyzed the answers and comment dynamics of over 6,000 questions with answers from Stack Overflow to understand how the commenting mechanism supports spreading knowledge. Throughout this study, the authors found that missing context, insufficient explanation, and code errors were common issues with these answers. To address these limitations, they suggested that developers supplement Stack Overflow answers with other sources of information, such as documentation, blogs, and textbooks. They emphasized the importance of utilizing various sources to understand software development issues comprehensively.

Furthermore, the study found that the quality of answers on Stack Overflow can vary depending on the experience level of the respondent. Experienced respondents provided more detailed and comprehensive answers, while less experienced respondents provided incomplete or incorrect solutions. This work showed that commenting behavior relates to user characteristics, such as experience level and previous social activities [43].

In another work on Stack Overflow, Sengupta et al. (2020) studied the content of users' comments to understand the role of comments and the community on Stack Overflow. The authors conducted a content analysis of over three million comments on the site. They found that comments served various purposes, including providing feedback, asking for clarification, and offering alternative solutions. Their study highlights the importance of comments as a source of information for learning on Stack Overflow, as they provide additional context and insights beyond the original question and answer. The authors identified several characteristics of comments that were particularly useful for learning, such as those that explained the reasoning behind a solution or pointed out potential pitfalls.

Moreover, the authors conducted a content-based analysis to categorize the comments into groups based on how they can support knowledge spread and learning. The study found that the Stack Overflow community was crucial in fostering learning through comments. The authors noted a supportive and collaborative community on the site, with users helping each other and providing constructive feedback [44].

Choi et al. (2015) investigated the content and user dynamics of conversations on Reddit using data mining techniques. The authors analyzed a large dataset of Reddit posts and comments to identify the critical properties of conversations, such as their length, style, and topic, and examine how these properties relate to user engagement and participation. The authors found that conversations on Reddit are characterized by a few highly active users who generate most of the content. In contrast, most users contribute only a small amount of content. This pattern of user engagement is consistent across different subreddits, although the level of user activity varies widely.

Additionally, the authors identified several types of conversations, such as debates, jokes, and question-and-answer sessions, and examined how users participate in each type. The authors' findings showed several implications for designing and managing online social networks. For example, they suggested that network operators should encourage user engagement and participation among a broader range of users to increase the diversity of content and viewpoints on the platform. They also highlighted the importance of understanding the different types of conversations that occur on social networks and developing tools and strategies to support them [45].

Social Roles

Understanding user roles in online social communities is an attractive area for researchers. Some research focused on determining a person's role within a more extensive network, while other works recognized roles in the specific communities in which the user interacts. Forestier et al. (2012) presented a survey article regarding identifying roles on social networks. They conducted a comprehensive literature review to explore the methodologies and research issues related to identifying social roles in networks. They proposed an analytical framework to study social roles and emphasized context's importance when identifying them. The authors argued that different networks have unique characteristics, which can influence the identification of roles. They suggested that researchers use a combination of methods to capture the complexity of social roles in networks.

The authors also discussed different roles identified in social networks, including influencers, which significantly impact other network members; bridges, which connect different parts of the network; and gatekeepers, who control access to the network. Additionally, the authors proposed a methodology in their paper to study social roles involving collecting data on user behavior and interactions and using social network analysis techniques to identify patterns and clusters within the network. By combining these methods, the authors argued that researchers could better understand individuals' roles within social networks and the evolution of these roles over time [46].

Gleave et al. (2009) presented a standard approach to defining social roles in online communities by combining social characteristics with behavioral and interaction-based activities. They proposed a conceptual and operational definition of "social role" in online communities. They drew on symbolic interactionism theory to define social roles as cultural objects recognized and accepted in a community and used to accomplish pragmatic interaction goals. They argued that these roles act as both constraints and resources for action and can be observed at different levels of analysis, such as communication content, identities enacted, local social network patterns, and behavioral history. The authors suggest that researchers infer the status of participants in online social spaces once these roles have been identified through detailed content analysis and a broader statistical analysis of structure and content. The authors also outline measurement and analysis strategies for identifying social roles in computer-mediated social spaces and demonstrate this process in Usenet and Wikipedia domains [47].

Onoue et al. (2013) conducted a study focusing on the activities of software developers rather than analyzing the social structure or roles within software projects. They selected two active software projects, node, and jQuery, hosted on the GitHub platform, to analyze the time developers spent on coding vs. commenting and issue handling. The study found that the top contributors in these projects exhibited varying characteristics of development activities, with some being specialists who worked primarily on their target project. In contrast, others worked on different projects or both. Although GitHub provides lists of top contributors based solely on the number of commits, their study concluded that a mix of developer types best served active projects. The authors also observed significant variations in developer activity rates, ranging from a few days to a year [48].

Understanding the roles of users on other social platforms attracted researchers. Buntain et al. (2014) studied Reddit communities to analyze users' behaviors to evaluate social communities' stability and provide recommendations based on users' roles in their social groups. They explored the use of network analysis to identify social roles within the online discussion platform Reddit. The authors analyzed a large dataset of comments and user connections on Reddit and used a variety of metrics to identify distinct social roles, such as moderators, contributors, and lurkers. The study found that different social roles had distinct activity patterns on the platform. For example, moderators were found to have a high degree of centrality within the network, while lurkers tended to have fewer connections and be less active. The study also found evidence of "bridge" users who connected different parts of the network. The authors argued that understanding social roles within online communities like Reddit is important for understanding how these communities function and designing effective moderation and community management strategies. The authors demonstrated that this role could be classified based on users' activities without needing content-based related analysis [49].

In another work to uncover the relationship between network users and their roles, Welser et al. [50] used different methods to visualize user interactions in Usenet communities. Using data from Usenet newsgroups, the authors identified distinct social roles based on patterns of communication behavior, such as frequency and type of messages sent and received. The authors focused on analyzing users who behave mostly to answer others within local communities. They then visualized these social roles using a network-based approach, highlighting the connections between individuals and their roles in the community. The authors demonstrated that their approach effectively identifies and visualizes various social roles, including experts, newcomers, and central connectors. They argued that their approach could provide valuable insights into the social dynamics of online discussion groups and could be used to inform the design of community management strategies. Their analytical work showed that users' activities within a community positively correlate to the engagement of the people who play the role of answer persons.

Code Review and Code Quality

Code quality is critical in software development; no one can deny its significance. It is one of the most crucial stages in the *doneness* of software. Online social networks like GitHub provide an excellent platform for researchers to study code quality and code review behavior in a naturalistic setting. Researchers can observe how developers collaborate and communicate on GitHub, and they can analyze the code and review comments to gain insights into the software development process.

Ray et al. (2014) conducted an exhaustive investigation using data obtained from GitHub to explore how language features affect software quality, explicitly focusing on the differences between static and dynamic typing and strong and weak typing. Using various methodologies and considering several confounding factors, including team size, project size, and project history, the authors aimed to understand the impact of language design on software quality. After analyzing the data, the study revealed that language design has a modest but statistically significant impact on software quality. Specifically, strong typing was marginally better than weak typing, and functional languages with static typing produced higher-quality software than those with dynamic typing. Additionally, functional languages demonstrated higher software quality than procedural languages. Despite these results, the authors cautioned that the observed effects could also be attributed to other factors, such as the preference of particular personality types for specific languages, which can be challenging to measure [51].

In a study by Yu et al. (2016), the authors investigated using GitHub's pull request mechanism to reduce potential bugs in software development. The study examined whether and how previous approaches used in bug triaging and code review could be adapted to recommend reviewers for pull requests while improving the recommendation performance. To this end, the authors extended three typical approaches in bug triaging and code review to the new challenge of assigning reviewers to pull requests by analyzing social relations between contributors and reviewers. Additionally, the authors proposed a novel comment-based approach to investigate developers' social communication before merging the code into the main codebase. Combining their new approach with traditional approaches and performing an analytical evaluation of all methods, the authors found that their proposed approach can perform similarly to traditional approaches when used alone. However, the study demonstrated that traditional approaches to code review and debugging bugs could be significantly improved by incorporating information extracted from prior social interactions between developers on GitHub. These findings confirm the feasibility of formal code review and debugging approaches while highlighting the potential for significant enhancements by integrating social communication analysis [52].

In a separate study, Yu et al. (2016) examined the influence of user rules on code quality and explored the possible relationship between user popularity and the introduction of code issues.

Their research involved a case study that aimed to assess the quality of code contributed by casual developers across some of the popular GitHub projects. In their results, the authors revealed that casual contributors introduced a greater quantity and severity of Code Quality Issues than main contributors. Furthermore, the study demonstrated no statistically significant difference in code quality performance between developers who contributed as main and casual contributors across different projects. Additionally, the authors found that casual contributors with fewer project stars tended to introduce more Code Quality Issues than their counterparts with more project stars [53].

McIntosh et al. (2014) investigated the association between code review coverage and reviewers' engagement and its impact on software quality. The authors conducted a case study of three git-based projects, namely *Qt*, *VTK*, and *ITK*, to assess the relationship between code review coverage, participation, and software quality. Their study showed a significant correlation between code review coverage and code reviewers' participation in software quality. Low code review coverage and participation were found to be associated with up to two and five additional post-release defects, respectively. These findings provide empirical evidence to support the intuition that poorly reviewed code can harm software quality in large systems using modern reviewing tools. The study highlights the importance of adequate code review coverage and participation in software development processes, as they may play a critical role in ensuring high-quality software products [54].

Visualization

Data visualization is essential for understanding complex networks, such as online social platforms. The massive amounts of data generated in online social networks can sometimes be challenging to comprehend without visualization techniques. In many studies, data scientists and researchers utilize data visualization techniques to provide a means of understanding the complexity of these networks and help identify potential areas for improvement.

Fiechter et al. (2021) presented a visualization technique for exploring and understanding issue-related behaviors in GitHub. The authors introduced a web-based tool called *GHVis* that visually represents the activity on a GitHub repository's issue tracker. *GHVis* allows users to explore the lifecycle of issues and their relationships with other entities, such as code commits, pull requests, and comments. The visualization techniques employed in *GHVis* enable users to detect patterns and trends in issue-related behaviors and to identify potential bottlenecks in the development process. Fiechter et al. conducted a case study to evaluate the effectiveness of *GHVis*. The results showed that it could help developers gain insights into the issue-related behaviors of a project and identify areas for improvement. The authors concluded that *GHVis* could be helpful for project managers and developers to monitor and improve the issue-tracking process in GitHub projects [55].

In their study, Liao et al. (2018) aimed to explore the characteristics of issue-related behaviors in GitHub, a widely used platform for software development collaboration. The authors used visualization techniques to analyze user behaviors related to issue creation, commenting, and closure in various repositories. They found that the number of issues created and closed varied significantly depending on the repository type, with some having a higher issue creation rate but a lower closure rate. They also identified different patterns of user participation in issue-related activities, such as a few active users dominating the discussion or a more evenly distributed participation among users. Furthermore, the authors found that users with different roles, such as contributors and maintainers, had different patterns of issue-related behaviors [56].

In another work, Celińska et al. (2017) proposed a unique visualization technique for exploring programming languages in GitHub. The authors leveraged the hyperbolic plane, a mathematical space that visually represents hierarchical data structures, to visualize the relationships between programming languages. They collected data from the repositories hosted on GitHub and mapped the programming languages into the hyperbolic plane, which they then displayed as an interactive visualization. The authors evaluated their approach by comparing the visualization results with existing taxonomies of programming languages. They found that their visualization approach provides a more informative and intuitive way of exploring the relationships between programming languages [57].

Dubey et al. (2018) explored data visualization techniques to analyze GitHub repository parameters. The authors used *ElasticSearch* and *Kibana* to extract and visualize data from various GitHub repositories. They analyzed the data to gain insights into the repositories' size, activity, contributors, and user engagement. The study found that data visualization techniques could help researchers better understand the trends and patterns in GitHub repositories [58].

Padhye et al. (2014) studied the external community contribution to open-source projects and analyzed over 1,000 popular projects hosted on GitHub. The authors identified the characteristics of external contributions, including the types of contributions, the frequency of contributions, and the impact of contributions on the project. They found that 75% of the projects received contributions from external contributors. They also observed that external contributors tended to make smaller, more focused contributions, such as bug fixes or new features, rather than significant code changes. The authors noted that external contributions played an important role in developing open-source projects, as they helped improve the software's quality and functionality [59].

CHAPTER 3: A POPULARITY-BASED MODEL OF THE DIFFUSION OF INNOVATION ON GITHUB

Most of the content in this chapter was previously published under the same title in the Proceedings of the 2018 Conference of the Computational Social Science Society of the Americas.

Introduction

Social media platforms have fundamentally transformed how information and ideas are disseminated and adopted. These platforms' speed and ease of communication enable ideas to travel rapidly worldwide. However, this also poses a challenge in distinguishing the actual transmission of innovation from the shared zeitgeist of an idea emerging from multiple sources. In the context of open software development platforms like GitHub, the infrastructure used for software version control can be repurposed to track the transmission of innovation from one software developer to the next.

Everett Rogers' seminal work on the diffusion of innovations identified several critical factors that influence the speed of adoption, including communication channels, time, and social systems [60]. Our research explores the impact of the GitHub social system on code adoption. With GitHub being one of the world's most prominent source code hosts, it comprises 24 million developers working across 67 million repositories [61]. The architecture of GitHub repositories allows users to commit their content and share it with others. Moreover, any GitHub user can contribute to a repository by sending a pull request, which the repository owner and collaborators can accept or reject. Contributors can also attach a comment to their commit or pull request to communicate a message.

Of particular interest for our study are three key event types that track public interest in a repository: forking, watching, and starring. Forking occurs when a user clones a repository and becomes its owner. Sometimes, forks are created by the original team of collaborators to manage significant code changes, but anyone can fork a public repository. Developers can watch a repository to receive all notifications of changes and star repositories to signal approval for the project and receive a compressed list of notifications. Forks are valuable for tracking the spread of innovation, and all three events (fork, watch, and star) have been used as measures of repository popularity.

To model the spread of innovation on GitHub, we introduce a popularity-based approach and use it to predict code adoption, as measured through fork events. Our model captures the relationships between GitHub repositories, users, and followers to generate event timelines and social structures. We analyze the impact of various social factors on code adoption, including the number of forks, watchers, and stars and the characteristics of the repository owner and collaborators. Our results demonstrate that incorporating popularity as a measure of social influence significantly improves modeling performance on *innovation adoption* and *sociality* dimensions. In this chapter, we contribute to the existing literature by providing insights into the role of social influence in adopting code on GitHub. Our findings have important implications for developers, managers, and policymakers interested in understanding the dynamics of innovation diffusion and social networks on software development platforms.

Method

Our research uses data from the GHTorrent project [62], an archive of public event data extracted using GitHub's REST API. There are 37 listed events on GitHub, including create, fork, watch, issue, push, and pull request events [63]. These events alert GitHub developers to changes in repositories that they are tracking.

Data Filtration

From the GHTorrent dataset, we only retained the fork events (41,014 in total). Each fork event object contains information about the user who generated the event and his or her followers. Additionally, it includes repository properties such as programming language, fork count, watchers, and size. The dataset is available for download on the *Mega platform* at `https://bit.ly/abdul-dissertation-dataset` under the folder **Chapter03DiffusionOfInnovation**. We created a copy of the dataset but simplified the fork event object only to include information relevant to our innovation diffusion model. A sample of our fork event object is shown below.

```
{
 'actor': 'GithubUser',
 'created_at': '2020-01-16T02:00:58Z',
 'followers':
  [
   'follower_1',
   'follower_2',
   'follower_3'
  ],
 'repo_created_at': '2017-01-01T17:44:32Z',
 'repo_forks_count': 4,
 'repo_language': 'C++',
 'repo_name': 'my_repo',
 'repo_network_count': 4,
 'repo_size': 536,
 'repo_stargazers': 2,
 'repo_subscribers_count': 4,
 'repo_watchers': 2
}
```

The programming language is relevant to code adoption. The total number of repository languages in our dataset is 160, where JavaScript was the most commonly used language in all the fork events. Table 3.1 details the programming languages of the repositories in our original dataset, ranked by the number of fork events.

Table 3.1: Top 10 repository languages, ranked by number of fork events

Repo Language	Fork Events
JavaScript	8,665
Java	5,925
Python	5,608
C++	2,366
HTML	2,278
PHP	1,717
Ruby	1,555
C	1,533
C-sharp	1,349
Go	1,180
Other languages	8,838

There are a few highly used languages, while many languages (aggregated in the table under the category Other Languages) only had a few fork events. To account for the dominance of the most common languages, we only retained the repositories coded in the top 20% (32 languages) and removed the rest. As a result, our dataset included 39,935 fork events, of which we randomly sampled 5,000 fork events to create our model.

Model Construction

Our model of innovation diffusion includes three components: 1) repositories, 2) actors (the users who fork the repositories), and 3) their followers. Since GitHub repositories host ideas, technologies, and methods, they can be considered knowledge resources used to transmit innovations among projects through the users and their followers. We link the components with three types of connections: *repo-actor*, *repo-repo*, and *actor-follower*. These connections naturally arise from GitHub's event and notification system:

- A GitHub user (actor) is able to create different events on multiple repositories.
- The followers of an actor are notified about commits made by their followee.

Thus, we conclude that a connection exists between a repository and an actor if an actor forks the repository. Likewise, the second interaction (repo-repo) implicitly occurs if a user creates two similar events on two repositories. These repositories are likely to be related to each other in multiple ways, including language, code structure, and topic. Also, there is likely a connection between two repositories that the same user forked. The actor-follower relationship is explicitly defined as the list of followers for a specific actor. Figure 3.1 illustrates a sample cluster in the network that contains all three connections.

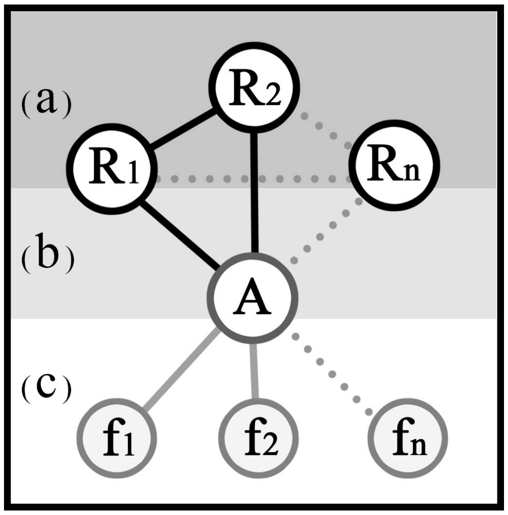


Figure 3.1: Our model contains three components (repos, actors, and followers) and three types of connections: (a) repo-repo (b) repo-actor, and (c) actor-follower.

Identifying Structural Changes over Time

Our model network spreads innovation by connecting repositories to potential adopters (followers). However, the network structure changes over time; to model this, we examine the structure at three checkpoints to study the knowledge transmission rate. Assuming that the initial structure is constructed at time t_0, three time points t_1, t_2, and t_3 are defined, where $t_0 < t_1 < t_2 < t_3$. The temporal distance that separates them is defined as a period during which a threshold number of additional fork events are created; our results use a threshold of 5,000 events between time points. The new fork events result in mergers of existing connections (repository-actor and actor-follower).

Once the followers are notified about a fork event creation by their followee, they may be eager to learn about the new repository by forking it. However, this is a probable interaction that could occur at some point in the future, where many other factors may be involved. Figure 3.2 shows the potential repo-follower interaction.

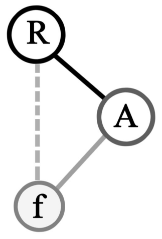


Figure 3.2: Repo-follower interaction: a probable connection that may occur when follower f of actor A forks repository R.

Knowing that the repositories do not possess the same features in our model, we assume that this fact affects the likelihood of a repository being forked by an actor's followers at a particular time. For instance, if there are two repositories (R_1 and R_2) where R_1 has higher popularity R_2, then we assume that it will fork at an earlier time than R_2.

Popularity is a feature that encompasses the repository's reputation. In this work, we use the popularity of the repositories as a relevant factor that identifies repositories' effectiveness at transmitting innovation. Out of the fork event object properties, the features that are correlated with the popularity of a repository are:

- ***repo fork count***: the total number of users that forked a repository
- ***repo watcher count***: the users that chose to receive emails or notifications about a repository
- ***repo stargazer count***: the number of users who bookmarked a repository

However, since a GitHub user who stars a repository is automatically set to watching that repository, in most cases, *repo_watching_count* and *repo_stargazing_count* are equal. By comparing the number of fork events and the number of watchers or stargazers of a repository, we realized that increasing one increases the other as well [64]. Figure 3.3 shows the correlation between repo fork events and watcher count.

Thus, to utilize repo popularity in our model, we use the following formulation for the probability of establishing a specific connection between repository R and a follower of the actor A who forked the repository R:

$$P(forkEvent_{t_j}(R, f_A)) = \beta\,[forkEvent_{t_i}(R,A)] \tag{3.1}$$

where:

- $t_i < t_j$,
- $forkEvent(R,A) = 1$, if A has forked R.
- f_A is a follower of A that has not forked R yet.

- β is the likelihood of becoming connected based on the popularity of R. β is closer to 1 if R is more popular than the other repositories. To find the value of β we compute the following fraction:

$$\beta = \frac{forkCount(R)}{\sum_{n=1}^{N} forkCount(R_n)} \tag{3.2}$$

where $forkCount(R)$= the number of fork events of R, and N= total number of repositories.

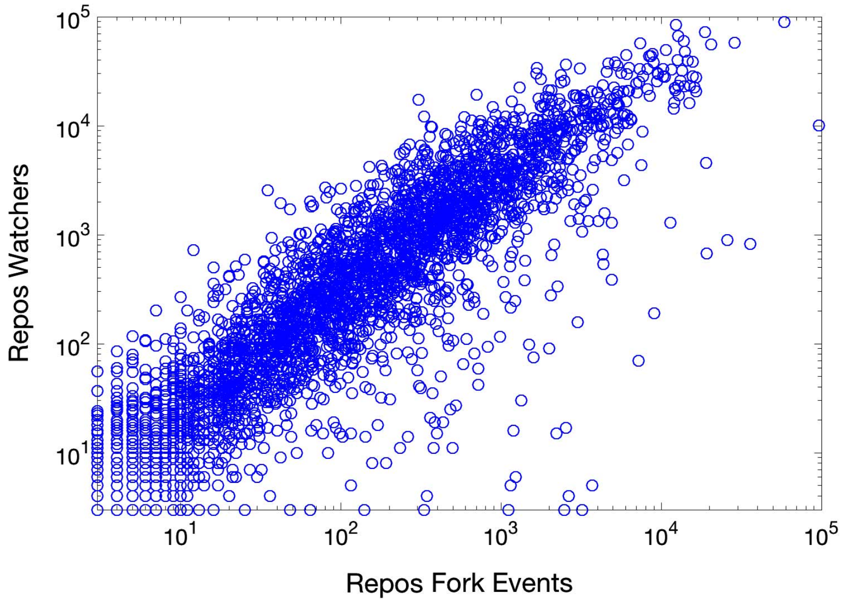


Figure 3.3: The correlation between the number of fork events and the watcher count of the repositories in our dataset

According to our assumption, once a follower forks a repository, the popularity of that repository will increase by one more fork event. At each time a connection may occur, a follower is selected randomly among an actor's followers to interact with the repository. Figure 3.4 depicts the evolution of a simple network generated using the example data provided in Table 3.2. We compare this model vs. a random network evolution model.

Table 3.2: A set of fork events is used to evolve the sample network (t_0) shown in Figure 3.4. The table presents a set of 3 events done by the actors A_1 and A_2 to fork the repositories R_1 and R_2. The table also shows two extra details: the followers of the actors and the fork event count for each repository.

Fork Event	Actor	Actor's Follower	Repo	Total Repo Fork Count
$forkEvent_1$	A_1	f_1	R_1	100
$forkEvent_2$	A_1	f_1	R_2	20
$forkEvent_3$	A_2	f_2	R_3	65

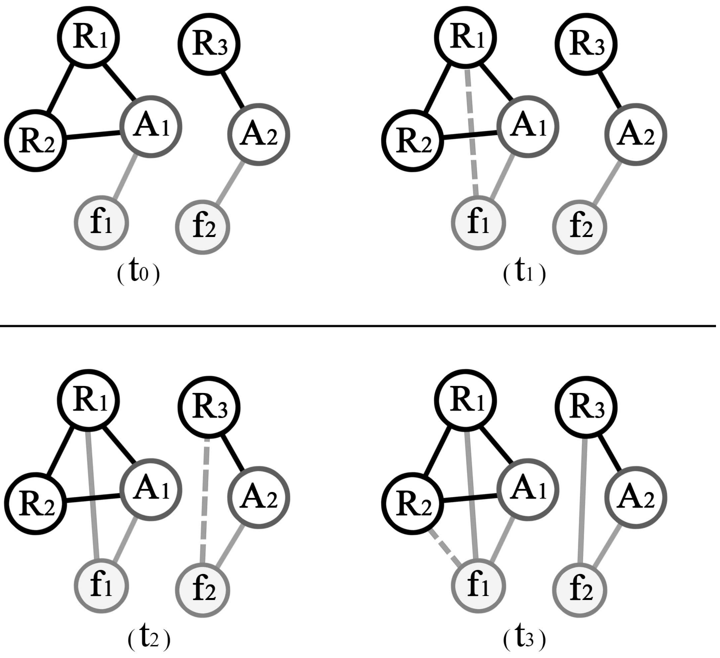


Figure 3.4: The popularity-based network evolution process is generated with the data in Table 3.2. (t_0) is the base model generated according to the data shown in Table 3.2; in (t_1) R_1 is the most popular repository, so the likelihood of it being forked by f_1 is higher. In (t_2) R_3 will have a higher chance of being forked and in (t_3) R_2. Notice that the temporal distance between the time points is the time required for one repository to be forked.

After designing a model that translates the existing relationships between repositories and users based on the occurrence of fork events, we used the data to create a network at the initial time point t_0 (Figure 3.5). Then we generate three versions of the network for the time points t_1, t_2, and t_3, where each one represents the structure with more introduced forks (repo-follower connections). Since we used repository popularity as a factor to compute the probability of adding fork events, we labeled these structures as *prob(ability) networks*.

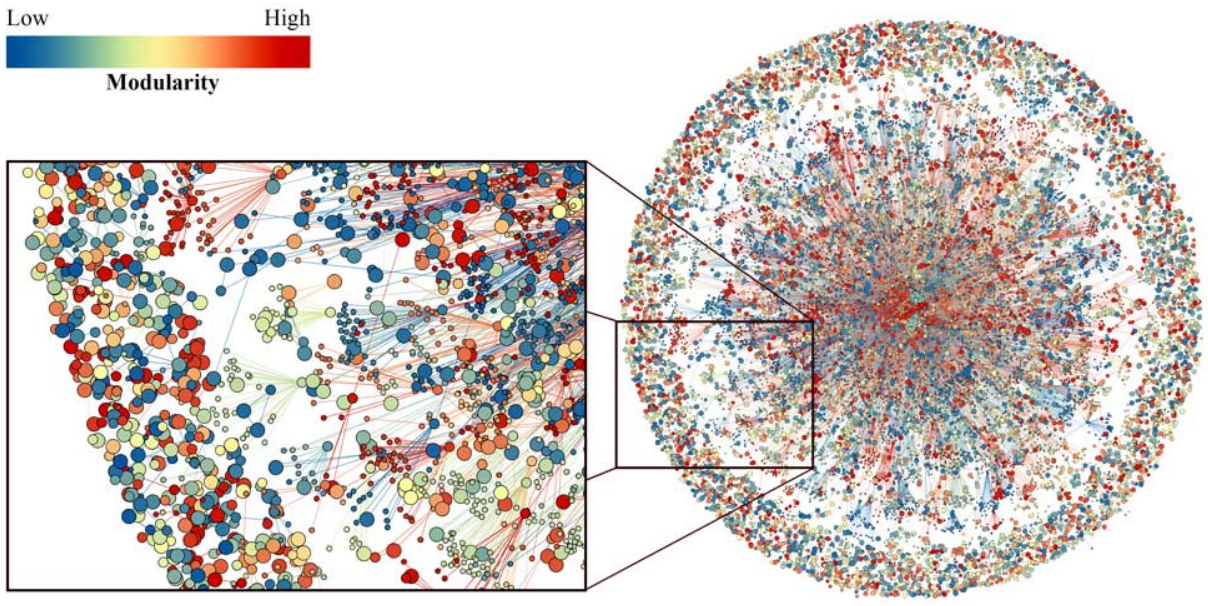


Figure 3.5: Shows the initial network of connections extracted from our GitHub fork event dataset. The colors indicate different communities in the network, which were detected with the modularity algorithm [1]. Red indicates higher connectivity among the nodes and blue marks less connected communities. The node size indicates the nodes' type (e.g., repository nodes' size > size of actor nodes > size of followers nodes). We used the open source software Gephi [2] to visualize this network (V=$19,250, E = 33,988$)

To evaluate the contribution of popularity, we compare the popularity-based model to a random network in which all repositories have an equal chance of being forked. Consequently, we ended up with one main network at t_0 (initialized with the data), and three more for each type (prob and random) representing the structures at t_1, t_2, and t_3.

The diffusion of innovation is defined as a multidimensional construct that seeks to explain the spread of knowledge across a population. Several factors affect the spread of ideas, techniques, and technologies. According to Rogers (2010), knowledge spread depends mainly on agents that act as intermediaries in this process [60]. Therefore, innovation must be adopted by more individuals to continue spreading over time and across the community.

In our experiments, we evaluate the popularity-based model for forecasting the diffusion of innovation, considering two dimensions: ***Innovation adoption***, which is measured by the spread rate of knowledge across a community, and ***Sociality***, which can be measured by the network's main structural properties and their effect on knowledge spread.

Innovation Adoption

To evaluate our models' characteristics, we compare the evolution of both models (popularity and random) at each time point compared to the original network. Figure 3.6 shows the cumulative distribution of fork events across repositories. The popularity model preferentially assigns events to repos with higher popularity, causing the popular repositories to become even more popular sooner. This behavior of the random models leans toward spreading knowledge more equally through repositories, independent of popularity. In the popularity model, the users who interacted with popular repositories can be considered potential early adopters who are more forward-looking than late adopters who do not follow the popular repositories. The evolution of the popularity-based model is similar to that of a preferential attachment network [65], where there is an increasing disparity in code adoption over time.

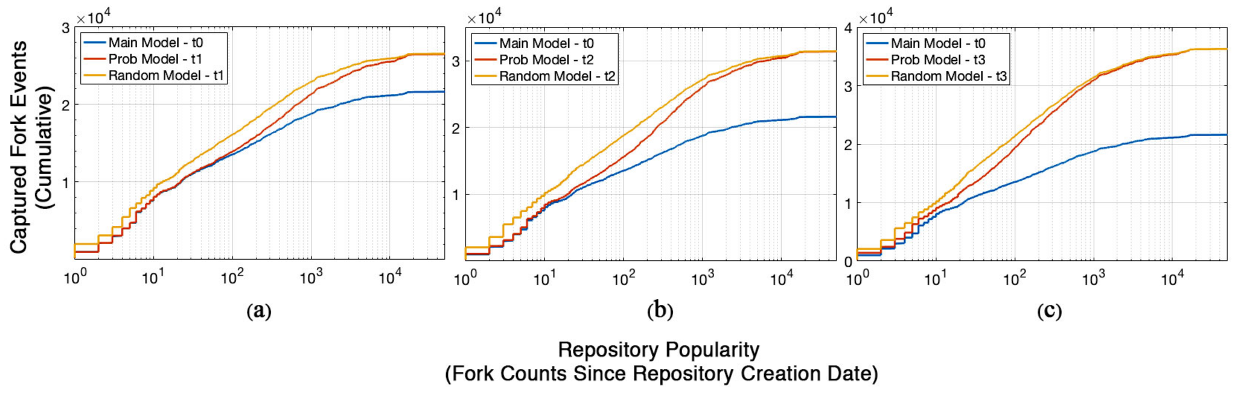


Figure 3.6: The repositories' rate of acquiring new fork events in the popularity and random models at three-time points, where the repository popularity is assumed to be the total number of fork events that occurred since creation. (a) depicts the difference between popularity and random structures at t_1 against the structure at t_0, while (b) shows the same comparison for the structures at t_2, as well as the comparison in (c) at t_3.

Sociality

One of the main elements that affect knowledge spread is the way a community is structured. According to Rogers (2010), communities with social characteristics will positively influence the diffusion of innovation [60]. We compare the network structure generated by our models using a set of standard network measures:

- ***Average Path Length***: indicates the average number of hops between two nodes in a network and is measured by the number of one-step links connecting them.
- ***Clustering Coefficient***: describes how tightly nodes are connected to one another.
- ***Degree Distribution***: the likelihood, $P(k)$, of possessing nodes with certain degree k.

Two common network classes are *small-world* and *scale-free* networks. Small-world networks possess highly connected clusters, resulting in a network with a high clustering coefficient [66]. Nodes can be reached in a few steps from anywhere in the network because of the lower average path length. Scale-free networks tend to have more nodes with low connectivity and a few highly connected nodes due to the power law degree distribution [67]. The degree distribution is a power-law [68] if it follows:

$$P(k) = ck^{-\alpha}$$

where the exponent value α falls in the range $2 < \alpha < 3$.

Table 3.3 compares the predicted network structure of both models at the three checkpoints. In most cases, the popularity-based model exhibits traits common to most social networks, such as shorter path lengths and higher clustering coefficients. However, we can see that the average degrees are the same since we allowed both model types to receive the same number of connections (fork events) for a more accurate comparison. Figure 3.7 shows that both model types have scale-free characteristics, meaning that the degree distributions fit the power-law; regardless of the time factor, the power-law exponent α falls between 2 and 3 for all cases.

Table 3.3: Characteristics of the evolving networks at different timesteps.

	Initialization	Popularity Model			Random Model		
	t_0	t_1	t_2	t_3	t_1	t_2	t_3
Average Path Length	5.75	5.59	5.49	5.42	5.60	5.51	5.44
Clustering Coefficient	0.53	0.73	0.80	0.84	0.74	0.79	0.82
Average Degree	3.51	4.03	4.55	5.06	4.03	4.55	5.06
Power-law Exponent α	2.00	2.04	2.07	2.08	2.41	2.12	2.08

Conclusion

Our research aims to understand the driving forces behind the diffusion of software innovations, as measured by GitHub code adoption over time. In this chapter, we examine repos popularity's role in code adoption. In essence, our model measures how predictive past fork events (popularity) are

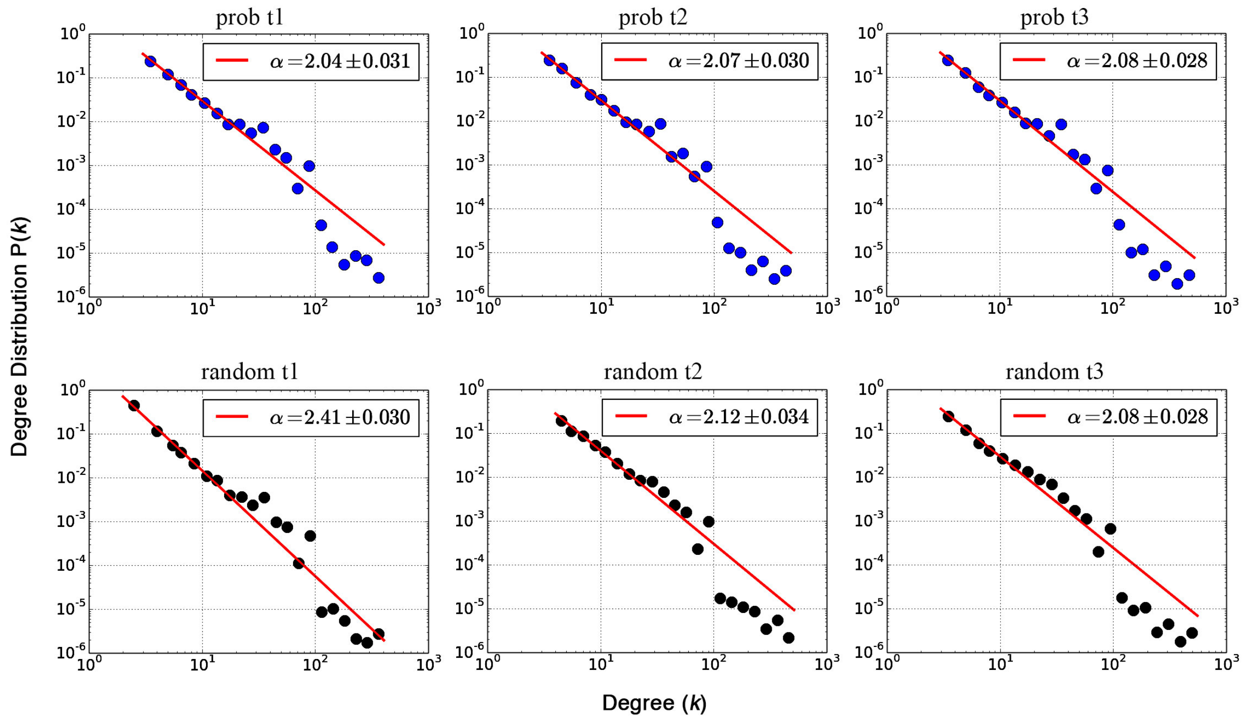


Figure 3.7: Power-law fitting of both popularity (prob) and random models degree distributions at all time points.

of future fork events (innovation).

To simulate knowledge spread, we compute the allocation of fork events across repositories and the network structure at three different time points after more fork events were introduced into the model from the dataset. Our results demonstrate that the popularity-based model is superior at modeling the main elements of diffusion of innovation: *innovation adoption* and *sociality*. Note that our model does not account for introducing new followers after repo fork events and uses a fairly simple measure of popularity.

Many GitHub event sequences are repetitive, with developers systematically committing code, reporting issues, and making pull requests. However, fork events are often more challenging to predict from past interactions, which may suddenly lead a software developer to become interested in a different code repository. The popularity-based model proposed in this section differs from other innovation diffusion models in that it focuses on the role of popularity rather than network structure. While other models may emphasize the importance of social networks and their interactions, this model treats network structure as a secondary effect [27] [28].

The popularity-based model of diffusion of innovation can help software developers, researchers, and experts better understand how communities of developers adopt new ideas and methodologies on GitHub in several ways:

- *Predicting future adoption*: The model can predict which repositories other developers will likely adopt based on their current popularity. This can help developers identify which projects are worth investing time and resources in and which may not gain traction.
- *Identifying influential developers*: The model can also be used to identify influential developers with many followers or who frequently contribute to popular repositories. These developers may have a greater impact on adopting new ideas and methodologies within the community.
- *Understanding social connections*: The model considers the social connections between developers, which can affect the spread of new ideas and methodologies. By understanding these connections, researchers and practitioners can develop strategies for promoting collaboration and knowledge sharing within the community.
- *Improving project visibility*: Developers can use the insights gained from this model to improve the visibility of their projects on GitHub by increasing their repository's popularity through various means, such as improving documentation, adding features, or engaging with other developers.

The insights gained here could be applied in other contexts, such as understanding how consumers adopt new technologies or products or how new ideas are spread within developer organizations. In addition to popularity, there are other features of the repositories, such as topic, the programming language, and the popularity of the contributors, which may also affect the future likelihood of repositories being forked. By understanding the factors that drive *innovation adoption* and *sociality*, we can develop more effective strategies for promoting the adoption of new ideas and technologies.

```
Data: Ground Truth Data
Result: GitHub network
for each input of GD do
    Create a new node;
    node.type = input.type;
end
for all nodes r where r.type = repository do
    a = the actor associated with repository r;
    Set link(r, a);
    if a.degree > 1 then
        for all other nodes m linked to a do
            Set link(r,m);
        end
    end
end
for all nodes a where a.type = actor do
    F = set of all followers of a;
    for all followers f in F do
        Set link(a,f);
    end
end
.
```

Algorithm 1: Constructing the model connections (repo-actor, repo-repo, and actor-followers) based on the ground truth data set GD that includes repositories with fork events *R*, actors *A*, and their follower set *F*

```
Data: Ground Truth Data
Result: GitHub network
1  Set NF = number of new fork events to add to PM;
2  for i = 1 to NF do
3      Calculate the probability of each repository in PM based on its popularity (Equation 3.1);
4      Set HR = the repository with a higher probability of getting another fork event ;
5      Set HA = Actor associated with HR;
6      Set HF = Set of followers of HA that haven't forked HR yet;
7      if HF is not empty then
8          Set f = a random follower from HF;
9          set link (f,HR);
10     end
11 end
12 .
```

Algorithm 2: Model update procedure for introducing fork events that have occurred since the last time point *PM*

CHAPTER 4: POPULARITY SCORE IN GITHUB

The majority of the material presented in this chapter was previously published as a paper in the 2020 International Conference on Computational Science and Computational Intelligence (CSCI), IEEE, under the title "Scoring Popularity in Github."

Introduction

GitHub is a leading social coding platform where users store their codes in repositories, share them, and engage in version control through various events. GitHub goes beyond version control, encouraging social interaction through actions such as starring content and watching repositories. These measures implicitly serve as a recommendation of the content's ideas, methods, innovation, and reusability. Being an author of a frequently forked repository is similar to being highly cited. GitHub has a gh-impact score that is comparable to h-index. Three different measurements of popularity are provided: *forks*, *stars*, and *watchers*. The *fork* count shows the number of people who were sufficiently interested in cloning or copying a repository on their local resource. In contrast, *stars* indicate users who have agreed to receive notifications about all repository activities. *Watchers* represent the number of users who bookmarked a repository.

Our study here reports on patterns and trends of popularity and social engagement in GitHub. Popularity measures make it easier for users to find highly recommended content rapidly. In the quest to achieve higher ratings, repository owners may be motivated to improve the quality of the hosted work to attract more people. The dark side of popularity ratings is that they may suppress users' creativity, leading them to remove unpopular content. For this reason, Instagram obscures the popularity of content to reduce users' anxiety. The following section discusses related work on popularity in social coding platforms.

In this chapter, we focus more on repositories' popularity. The aim is to understand, study, and evaluate the characteristics of popular repositories. Most similar works and applications usually use one of the forks, stars, or watchers to represent a repository's popularity. However, only considering one of these measurements may not be an overall indicator of popularity. In this work, we investigate and understand the concept's popularity over time and propose a unique measurement to indicate popularity accurately.

Method

Dataset

To prepare our dataset, we utilized the online archive of public repositories by the project GHTorrent [62], which extracts all the data using GitHub's REST API, and we captured 36,000 public repositories. Each record contains a repository object, including all the endpoints of the related events, commits, and contributors. The data objects also provide repository features and information such as the creation date, the primary programming language, the repository size, fork count, watcher count, star count, and repository owner information. Figure 4.1 presents the distribution of the critical properties of our dataset. Figure 4.2 shows a sample of the extracted information as a JSON object. The dataset is available for download on the *Mega platform* at `https://bit.ly/abdul-dissertation-dataset` under the folder **Chapter04PopularityScore**.

In this research, the main focus will be on the provided popularity indicators forks, watchers, and stars, and then extract their timeline of occurrence since the creation of the repository. To extract the timeline of occurrences for popularity-related events, we used the property *timestamp* provided by GitHub API to retrieve the exact time points that a fork or star event occurred.

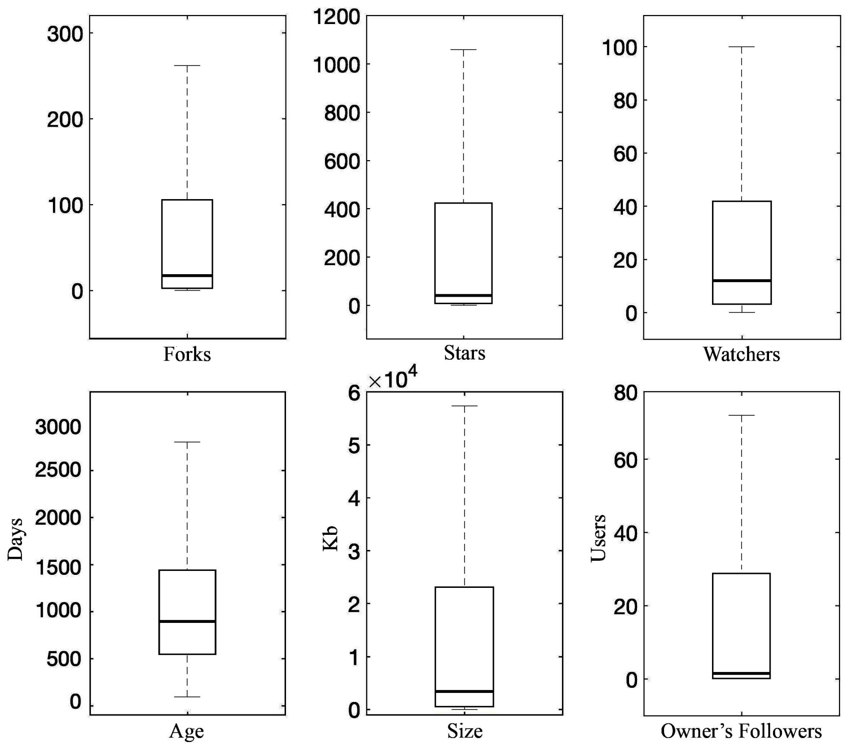


Figure 4.1: Distribution of the repositories features: forks, stars, watchers, age of repositories, size, and the number of repository owners' followers.

```
{
 'repo_created_at': '2016-05-11T13:04:39Z',
 'repo_name': 'my_repo',
 'repo_owner': {
      'owner_name': 'Bob',
      'repo_owner_follower_count': 312
 },
 'repo_language': 'Python',
 'repo_url': 'https://api.github.com/repos/Bob/
     my_repo',
 'repo_size': 1761,
 'repo_forks_count': 312,
 'repo_stargazers_count': 2109,
 'repo_watchers_count': 810
}
```

Figure 4.2: Presents a sample JSON object that represents the needed information for each repository, which was extracted from the collected dataset.

Popularity Growth

Three of the repository features on GitHub can be indicators of the repository's popularity: forkees, stargazers, and watchers.

The fork count shows the total number of times GitHub users cloned a repository. Many see it as an indicator of users' interest in others' works, innovations, and methods. It is more likely to see innovative solutions and advanced approaches attract more developers than non-original and uncreative ones. Therefore, the forks count is a good indicator for measuring the innovation popularity on GitHub.

Stars count represents the number of users who have bookmarked a repository or, as GitHub notes, the total number of repository stargazers. GitHub developers that *star* a repository can be seen as those who have found interesting hosted projects to bookmark. When a developer *stars* a repository, it will be easier to discover projects with similar topics in the future. Also, all the starred repositories can be located in one location on their GitHub account. That made staring handy and practical and may interest developers to star more repositories; the fact that interprets why most of the GitHub-related studies and applications have used stars as the primary indicator for repository popularity.

On the other hand, watch measurement is similar to the star feature when following other developers' projects, but with a slight difference. By watching a repository, users agree to receive notifications when a new event is created or a new developers discussion occurs for the watched repository.

The concept of popularity cannot be separated from time. For instance, talking about repositories' popularity growth becomes meaningful when it happens over a period of time. Eventually, repositories gain more popularity over time, which might be much faster for some and slower for others.

Figure 4.3 depicts the correlation between repository age (in days) and popularity indicators, in which we see an increase in the total number of people who fork, star, or watch a repository.

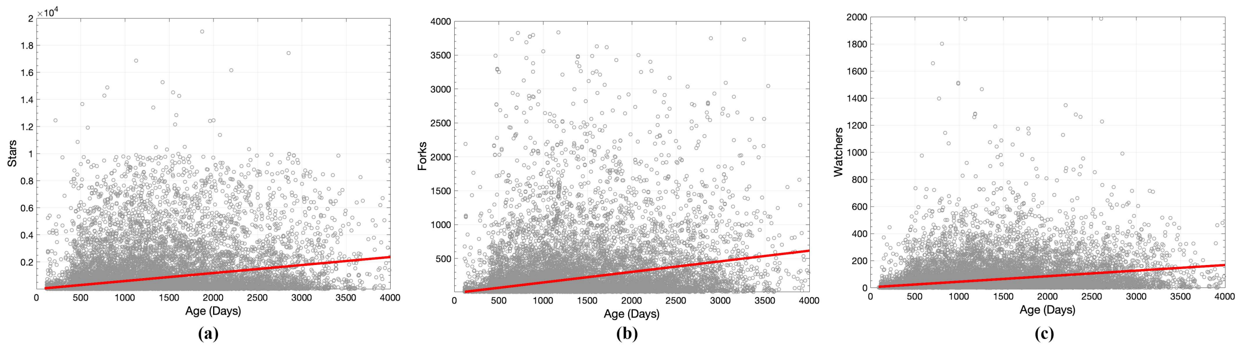


Figure 4.3: Correlation of age and popularity measurements. (a) shows age-stars correlation with a coefficient of 0.3215. (b) shows the age-forks correlation with a coefficient of 0.2923, and (c) shows the age-watchers correlation with a coefficient of 0.3162 [3]

.

Usually, repositories experience a spike in popularity growth. For instance, the repositories created by big tech companies and well-known organizations may attract developers in a single burst. Figure 4.4 demonstrates an example of such a repository, which received more than 9% of its total stargazers count in February 2017 and was created in October 2016.

Repositories can be grouped into three categories based on their popularity grew over time. The first group includes the repositories that have only attracted a small number of developers. They fail to raise their popularity to a noticeable point and remain at the same level for a long time. Consequently, this type of repositories may experience little growth. Some repositories gain popularity via various means but then lose it when some of GitHub users *unwatch* or *unstar* them. The last group consists of repositories that keep gaining popularity, where their overall growth rate is positive over most of their lifetime. These repositories may be considered as resources of innovation for developers on GitHub [35].

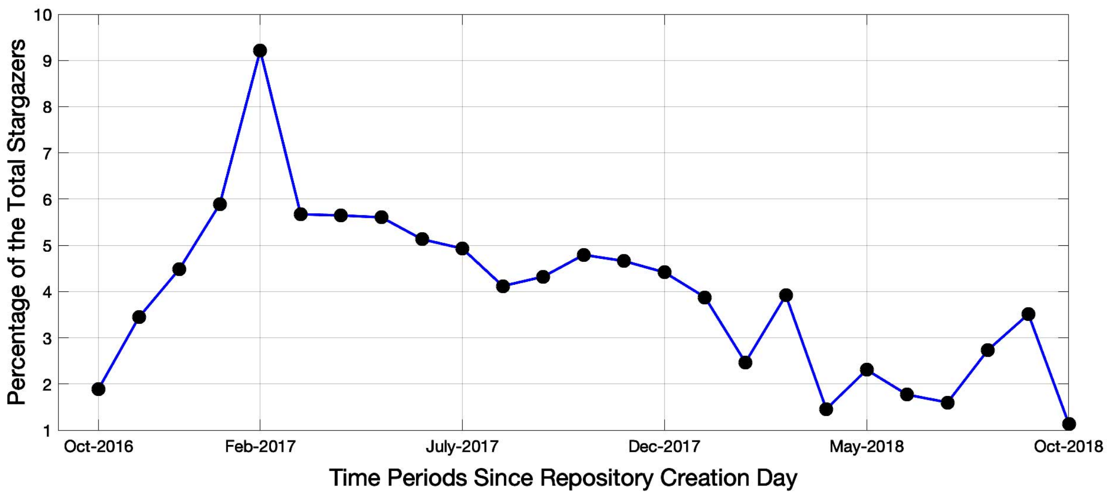


Figure 4.4: Shows the *stargazer's* growth of a randomly picked GitHub repository.

Calculating Popularity Score

Popularity in GitHub can be viewed as a measurement of how attracted users are to the repository content. The number of watchers, stargazers, or forkees may only partially capture this aura of attractiveness. We propose using a weighted popularity score to express different popularity aspects more accurately.

We illustrate the benefits of other scoring techniques with an example. Imagine there is a repository (A) that is forked by 1000 users and starred by 20 and a repository (B) that is not forked at all yet but has been starred 1020 times. Considering star count as the popularity indicator, repository (B) is more popular than (A), but looking at only fork counts, repository (A) is more popular than repository (B). However, it is helpful to consider time as a factor and to examine how repositories gained forks and stars. Some may be more stable in gaining popularity, and others may fluctuate over time, gaining thousands of stars/forks in a month and a few stars/forks in another month. A weighted score can consider all these factors.

Popularity Weight

Popularity is a term that gets features from interactions between a community's components. It is almost a meaningless concept when considering an element in isolation. A set of repositories can be considered a community in which the popularity of a repository is related to other repositories. In other words, a repository that receives more stars, watchers, or forks has impacted others' popularity. The idea of time-based popularity weights emerges from the fact that we see thousands of developers forking, starring, or watching different repositories during some periods. In contrast, the number of popularity-related events is much lower at other periods. Hence, we look at these periods as having different popularity weights. Consequently, gaining forks, stars, or watchers at certain times may be more valuable than in other periods.

We introduce the *Weighted Total Popularity Score (WTPS)* to calculate the popularity of a repository that is part of a community in which the popularity is weighted concerning time (See Figure 4.5).

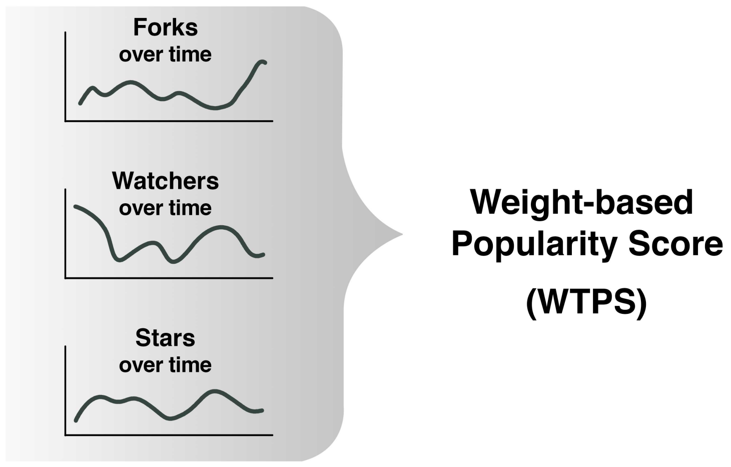


Figure 4.5: The popularity score is extended from the combination of some of the main popularity indicators of a GitHub repository.

Initially, the time interval t is equal to one month. Therefore, to find the popularity indicators' weights, we take all the forks and stars captured each month, and then we find their weights against the total gained forks and stars of the repositories of our dataset. Weight of forks (W_{F_t}) and weight of stars (W_{S_t}) for each time interval t are calculated as follows:

$$W_{F_t} = \frac{\sum_{i=0}^{n} Forks[R_i]_t}{\sum_{i=0}^{n} Forks[R_i]} \tag{4.1}$$

$$W_{S_t} = \frac{\sum_{i=0}^{n} Stars[R_i]_t}{\sum_{i=0}^{n} Stars[R_i]} \tag{4.2}$$

such that $\sum_{i=0}^{n} W_{F_t} = 1$ and $\sum_{i=0}^{n} W_{S_t} = 1$, where n is the total number of the repositories. Note that to compute the popularity score of an individual repository because it is part of a larger community of repositories on GitHub is equivalent to setting W_{F_t} and W_{S_t} to be equal to 1. Table 4.1 details a sample dataset of repositories and their captured forks and stars at different time intervals, along with their calculated W_{F_t} and W_{S_t} values. We can calculate the weighted total popularity score $WTPS$ for each repository R by calculating the forks and the star's weight at each time interval.

Table 4.1: A sample dataset of four repositories and their collected forks and stars at 5-time points [t_1... t_5] and their calculated weight W_F and W_S using Equations 4.1 and 4.2

	F_{t_1}	F_{t_2}	F_{t_3}	F_{t_4}	F_{t_5}	F_{total}	S_{t_1}	S_{t_2}	S_{t_3}	S_{t_4}	S_{t_5}	S_{total}
R_1	15	20	3	9	7	54	6	12	2	15	10	45
R_2	10	10	5	3	30	58	6	5	6	5	8	30
R_3	5	8	4	10	15	42	1	3	4	22	20	50
R_4	14	12	10	5	5	46	12	14	10	13	6	55
Weight	$\frac{44}{200}$	$\frac{50}{200}$	$\frac{22}{200}$	$\frac{27}{200}$	$\frac{57}{200}$	1	$\frac{25}{180}$	$\frac{34}{180}$	$\frac{22}{180}$	$\frac{55}{180}$	$\frac{44}{180}$	1

Table 4.2: Calculating the Weighted Total Popularity Score ($WTPS$) for the repositories detailed in Table 4.1 based on Equation 4.4

$WTPS$	R_1	R_2	R_3	R_4
t_1	4.08	2.98	1.23	4.64
t_2	7.25	3.44	2.56	5.63
t_3	0.57	1.28	0.92	2.32
t_4	5.71	1.9	7.93	4.57
t_5	4.43	10.5	9.15	2.89
$total$	22.04	20.1	21.79	20.05

Weighted Total Popularity Score ($WTPS$)

The equation below shows how to calculate the $WTPS$ of a repository R at time t:

$$WTPS[R_i]_t = (Forks[R_i]_t * W_{F_t}) + (Stars[R_i]_t * W_{S_t}) \tag{4.3}$$

Hence, by summing up all the calculated weighted popularity scores of a repository for all the time intervals, we will be able to obtain the overall $WTPS$ as follows:

$$WTPS[R_i]_{overall} = \sum_{t=0}^{m} WTPS[R_i]_t \tag{4.4}$$

where m is the number of the time intervals. Table 4.2 shows the $WTPS$ of each of the repositories of Table 4.1 and at each time point $t1, t2, ...$ as well as the $WTPS_{Overall}$ where both forks and stars are utilized to find the overall popularity score.

Results

This section presents a comparison of the use of $WTPS$ as an indicator of repository popularity vs. the standard measurements.

Ranking Popularity

The previous section described repositories with different fork/star ratios at different time points. Using our proposed approach of extracting their popularity, we calculated the popularity score $WTPS$. Figure 4.6 depicts a sample comparison of the repositories detailed in Table 4.1 based on their forks count, stars count, and the calculated WTPS.

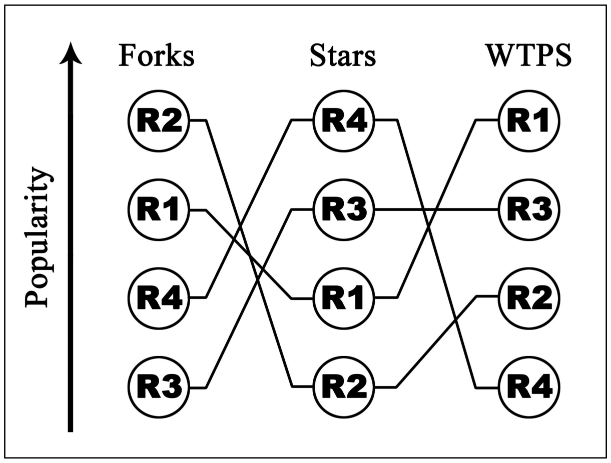


Figure 4.6: The rank of repositories of Table 4.1 based on their popularity indicator forks, stars, and $WTPS$.

By comparing a repository's forks and stars against its $WTPS$ we can find that its popularity level changes. When we consider the number of the accumulated forks or stars of a repository as the only popularity indicator, repositories may be ranked either lower or higher than when their popularity

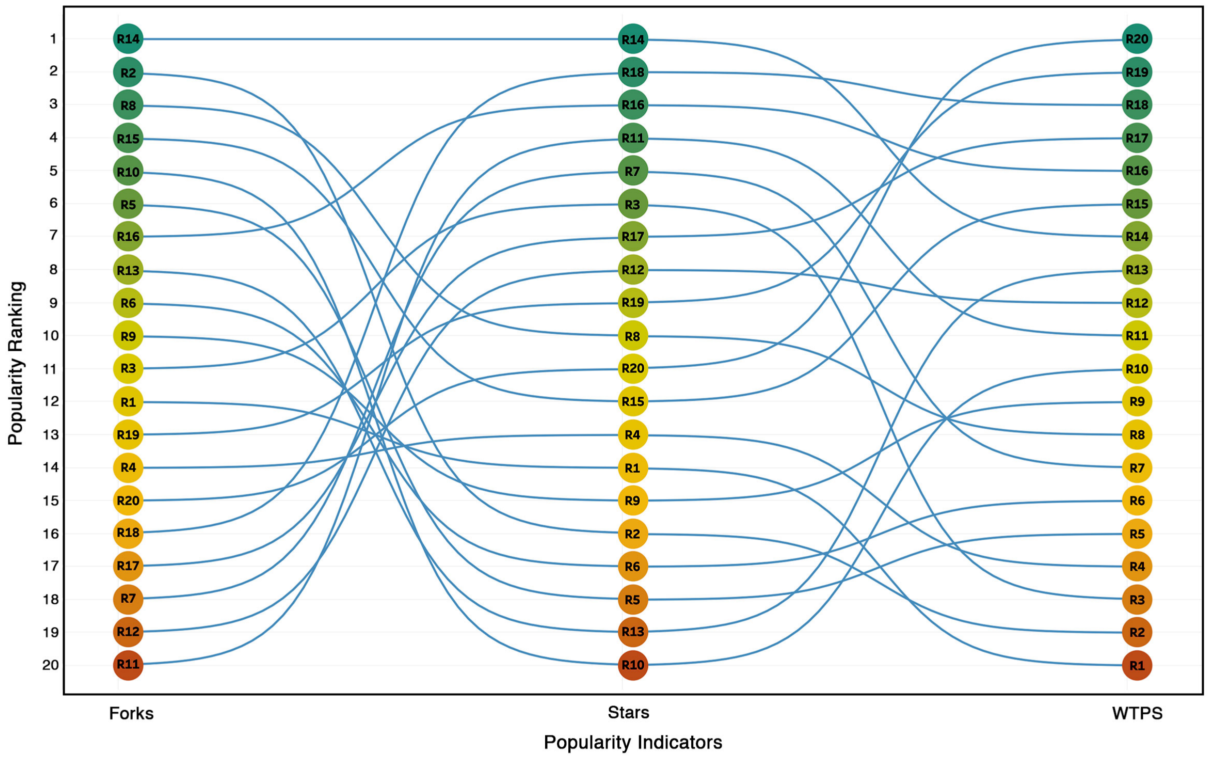


Figure 4.7: A sample of 20 repositories' popularity ranking using different popularity measures: forks, stars, and $WTPS$ (t= one month).

score is based on other measurements. In other words, if a repository looks popular among other repositories based on the number of the gained forks, it may or may be at a different level of popularity when star count is the popularity indicator. Figure 4.7 depicts repository rank changes based on the popularity indicator. For instance, even though R_{14} has the highest ranking among other repositories in fork counts as well as the star counts, it does not have the highest $WTPS$. Since $WTPS$ reweights forks and stars based on their relative standing at different periods, this results in a lower popularity score for R_{14}. This means that at some points, R_{14} gained forks or stars at a time when the popularity weights were relatively lower. Below, we utilize a linear regression approach [3] to analyze the correlation between $WTPS$ and other repository properties in more detail.

The proposed approach of calculating the popularity score, $WTPS$, is related to forks and stars; however, these popularity measurements will have different weights during different time periods. For instance, a repository can gain a large number of stars compared to other repositories at a certain point in its lifetime, which will lead to a greater total popularity score. To see the overall effect of fork count on the value of the $WTPS$ against the total count of stars gained by a repository, we compare their correlation to the total $WTPS$. Figure 4.8 shows how $WTPS$'s value correlates to repository fork and star counts. It is noticeable that the increase in the number of forks generally leads to a greater $WTPS$ value. The same statement validates the star counts and the popularity score value.

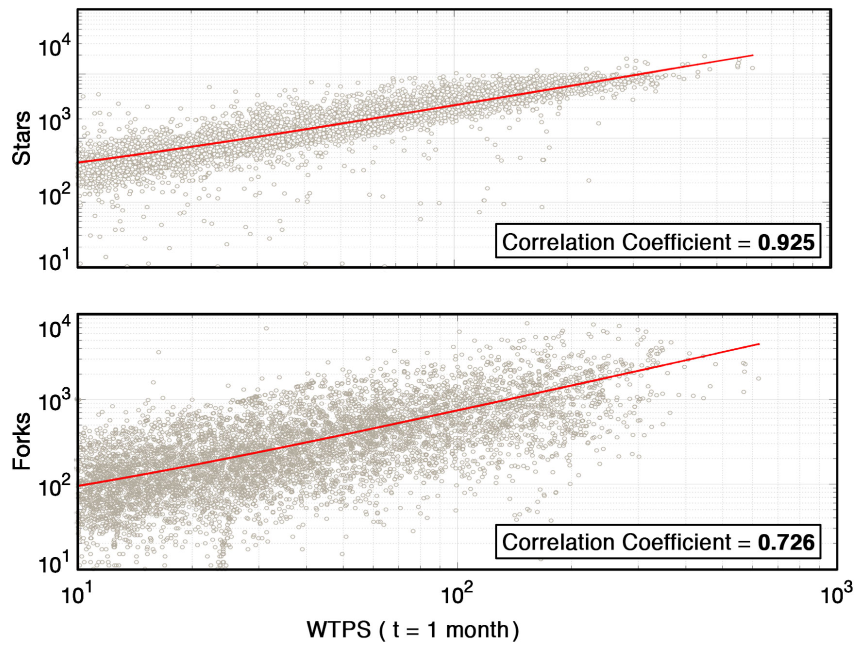


Figure 4.8: WTPS-Forks and $WTPS$-Stars correlation.

Even though they both have a strong relationship with the value of $WTPS$, the star counts are more correlated with the general popularity score values of repositories. For our data set, the star count correlation to $WTPS$ is 0.925, which explains that repositories with more stars are likely to have a high $WTPS$ value. This is aligned with the majority of research that uses the star count of a repository as an indicator of its popularity. Table 4.3 shows the correlations; it is clear that there is not a strong relationship between age, size, and owner's followers count and the popularity score $WTPS$. However, the age correlation to $WTPS$ is higher than other metrics, which may cause the time to have some effect on the popularity score.

Table 4.3: Correlations of $WTPS$ and repository properties

Repository Metrics	$WTPS$
Age	0.236089
Owner Followers	0.136390
Size	0.000755

In order to find the best time interval length, three more time interval lengths (in addition to 1 month) were defined: one week, two weeks, and three weeks. To do this, we computed the new $WTPS$ correlation to other popularity measurements. Table 4.4 details the results in which the time number of repository watchers was included. $WTPS$ correlation to forks, stars, and watchers are similar, with a more positive slope for shorter time interval lengths. However, in general, there will not be a massive difference in the value of $WTPS$ when selecting time intervals as small as one week and as large as one month.

Table 4.4: The calculation of a regression line for popularity indicators forks, stars, and watchers and $WTPS$ based on four different time intervals.

	Forks		Stars		Watchers	
$WTPS$	Slope	Coefficient	Slope	Coefficient	Slope	Coefficient
t = 1 month	7.259	0.726	32.267	0.925	1.728	0.691
t = 3 weeks	10.595	0.727	47.311	0.929	2.518	0.688
t = 2 weeks	15.722	0.72	70.129	0.928	3.744	0.689
t = 1 week	31.142	0.726	138.961	0.928	7.411	0.689

Popularity in Collaboration Networks

Earlier, we discussed that repositories might rank higher or lower when considering different popularity indicators. Hence, if we treat the community of the GitHub repositories as a network of repositories, it will exhibit different behaviors based on how repositories are weighted. A more popular repository may have a more significant impact on the characteristics of this network than the less popular one.

We consider different versions of the same network to understand the effect of selecting different popularity indicators on community characteristics. The model defined here has two types of nodes: repositories and the followers of repository owners, where each repository is linked to its owners' followers. Figure 4.9 presents a sample subset of the model; as it is shown R_1 is linked to its owner's followers $\{f_1, f_2, f_4\}$, R_2 is linked to the followers $\{ f_2,f_3,f_4 \}$ and R_3 is connected to $\{f_2, f_3\}$. Hence, considering the model described above, we constructed a graph of repositories and their followers. Figure 4.10 shows the graph where the large-sized nodes are those repositories with higher $WTPS$. However, a repository node size can differ depending on the selected popularity indicator.

In this section, the aim is to determine whether selecting different popularity indicators has a similar behavior on the graph's structure or not. For this purpose, we evaluate the impact of

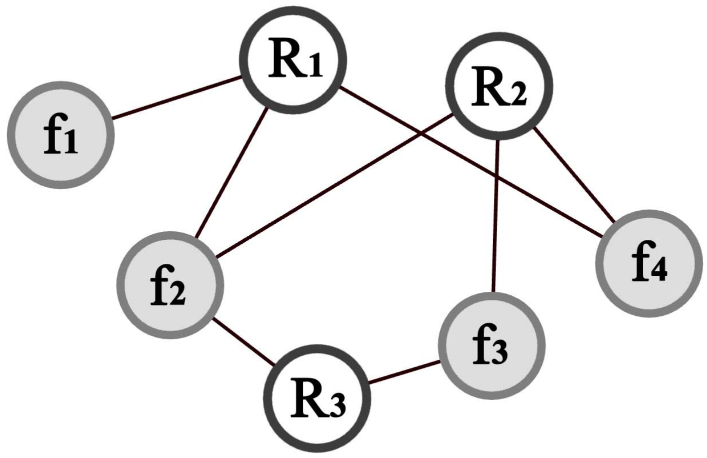


Figure 4.9: Subset of an example network: repository, owners' followers, and the repository-follower connections.

deleting nodes according to different popularity measures on the total clustering coefficient of the graph. Initially, the total clustering coefficient of the graph is calculated, and then, on each step, we remove a node from the graph that has the highest popularity among others. At each step, we select a node that is supposed to be removed based on different popularity measures: highest $WTPS$, highest stars, highest forks, and highest watchers, and at each time, the clustering coefficient of the graph is calculated.

As shown in Figure 4.11 the clustering coefficient value becomes smaller after every deletion. This behavior is similar for all four types of popularity-based node deletions. However, the trend is more similar when the node selection is based on the highest $WTPS$ or stars. Deletion based on the number of watchers leads to a slightly different result.

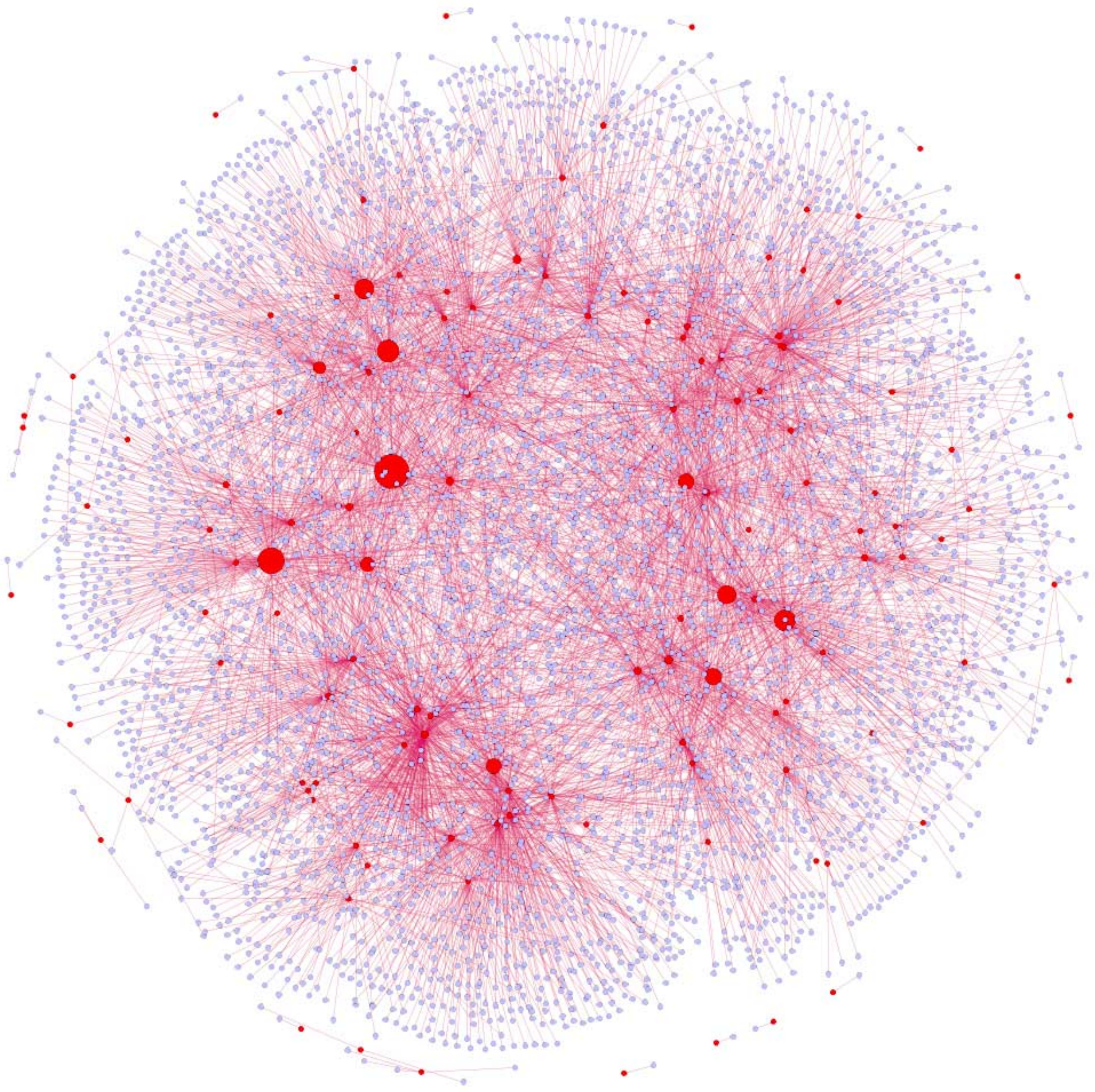


Figure 4.10: A sample graph of 5000 nodes and 5195 edges, where the red nodes represent the repositories and the blue ones are the repository owners' followers. The size of the repository nodes also is related to their popularity. This graph used $WTPS$ as a measurement of the popularity of the repository. We used the open-source software Gephi [2] to visualize this network.

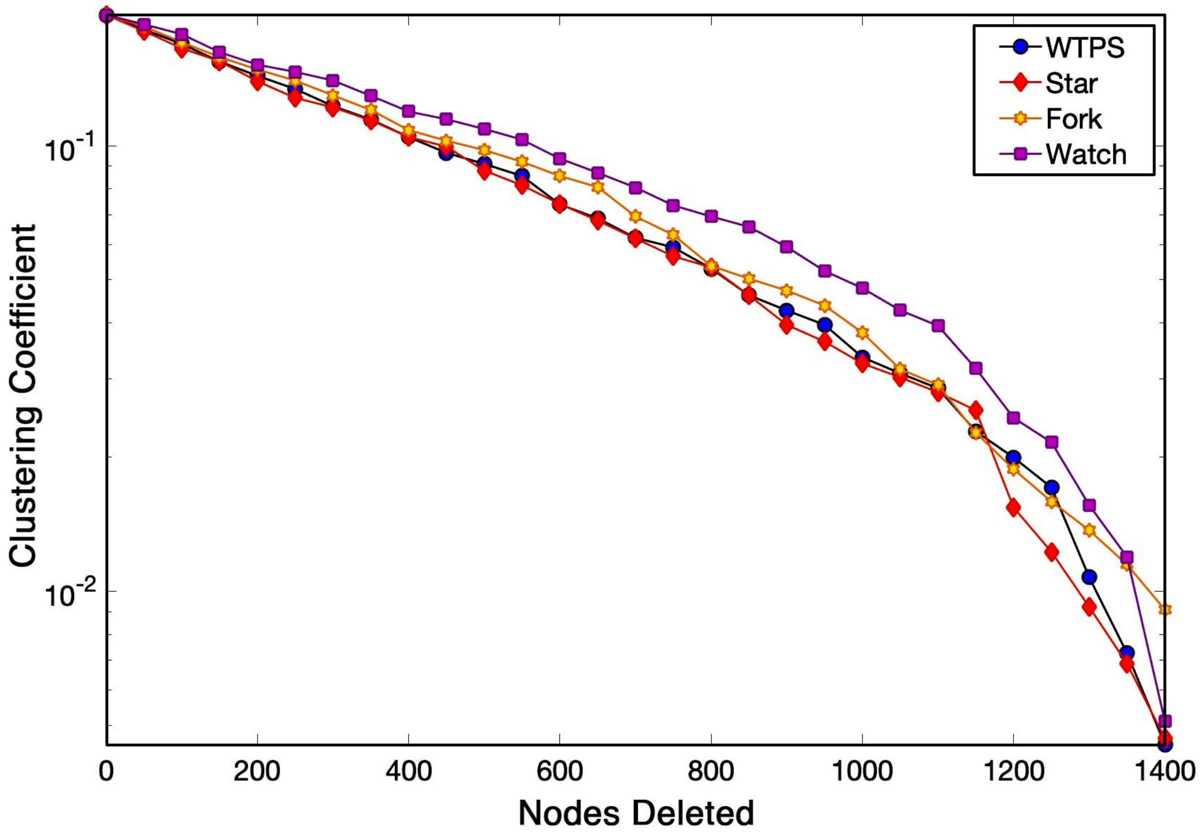


Figure 4.11: Repository deletion impact on one of the principal network characteristics: clustering coefficient.

Conclusion

The popularity of GitHub repositories has been a topic of significant interest in the software development community. To contribute to this field, this study aimed to propose a general popularity score for GitHub repositories and examine its correlation with existing popularity indicators, such as forks and stars. Our proposed popularity score, the Weighted Total Popularity Score ($WTPS$), is a weight-based metric that considers the gained forks and stars with different weights at different times during the repository's lifetime.

To ensure the validity of $WTPS$, we needed to develop a community of GitHub repositories to calculate the weights, as the weight of a repository's popularity at a specific time is defined as the ratio of its gained popularity within that period over the total number of gained popularity of all other repositories at the same period. We collected an extensive dataset of GitHub repositories using the GitHub API and calculated $WTPS$ for each repository.

The analysis of the correlation between $WTPS$ and other popularity metrics showed that $WTPS$ is significantly correlated with both forks and stars. However, the correlation analysis also revealed that $WTPS$ is more closely related to the number of stars than forks. This finding implies that a higher value of $WTPS$ corresponds to a greater likelihood of the repository garnering a more significant number of stars rather than forks, thereby increasing its overall popularity score. In other words, our study indicates that gaining more stars on a repository is more likely to enhance its popularity score than gaining forks.

Moreover, we performed a correlation analysis between $WTPS$ and other repository measurements to explore potential factors that may influence the popularity of a repository. We found low correlation coefficients between $WTPS$ and age, the total number of followers of a repository's owner, and the repository's size. However, age had the highest correlation with $WTPS$ ($\sim$ +0.27) among these three factors. Although the correlation is not strong, it can mean that the older a repository is, the higher the likelihood of it having a greater $WTPS$.

In addition, we constructed a graph that represents the relationships between GitHub repositories and their owners' followers. By linking a follower of a repository owner to the repository with an edge, we studied the impact of removing popular repositories one by one from the community, with each deletion resulting in the retrieval of the graph's total clustering coefficient. We conducted the popularity-based deletion process with $WTPS$, stars, forks, and watchers to compare their impact on the graphs. The results of node deletions for $WTPS$ and stars were the most similar, indicating that both metrics play a critical role in the popularity of a repository.

Finally, to facilitate further research in this area, we have publicly made our dataset and the code for scoring repositories available. The findings of this study have important implications for developers, software engineers, and data scientists seeking to optimize their GitHub repositories' popularity and visibility in the GitHub community. Overall, this study contributes to the ongoing discussion on GitHub repositories' popularity and provides a novel approach to measuring popularity with $WTPS$.

CHAPTER 5: IMPROVING CODE QUALITY IN GITHUB

The primary content of this chapter was previously published as a paper in the 2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, under the same title.

Introduction

Quality is the degree to which a software implementation fulfills specifications and customer expectations. However, there is no uniform definition of code quality since developers have various ideas about what constitutes excellent code. When analyzing software and determining its quality, there are a few different viewpoints to consider [69]. Almost everyone believes code quality is crucial, regardless of how we define it. While there is no one technique to assess the code's quality, there are a few key factors to consider in order to have a better understanding of the code's quality:

- *Extensibility*: the software's ability to accommodate future expansion.
- *Maintainability*: the ability to continuously modify the code while also controlling the risks that come with it.
- *Readability and Clarity*: the ability to adhere to language-specific standards, ensuring the application's structure is consistent, transparent, and straightforward.
- *Documentation*: supports other developers in comprehending the code and digesting its complexities in preparation for future development.
- *Testability*: the degree to which the program facilitates testing. It is mainly determined by a developer's ability to design, execute, automate, and control every test case.

- *Efficiency*: evaluating a program to see if it just consumes the necessary resources. It also assesses how well a code performs in the shortest period of time.
- *Reliability*: measures the chance of a code functioning without failure for a certain amount of time.
- *Portability*: measures the independence of a code in relation to the environment.
- *Reusability*: to determine the number of inter-dependencies to reuse the code.

Code review is one of the best approaches for improving the overall code quality. The quality measures mentioned above may not be feasible if the code is not adequately reviewed. GitHub's *Issue tracking* system provides a process to manage the code base bugs, general project tasks, and the necessary action items collaboratively. They are essential for the developers to collaborate, communicate, review the codes, and track defects. The overall flow of the GitHub Issue Tracker system is depicted in Figure 5.1.

Any GitHub user may start an Issue and moderate it. They have a discussion forum where they can keep track of work, ask and answer questions, share expertise, and disseminate new ideas. Users can attach one or more pull requests to Issues for the team to evaluate, identify, and defect before approving the new proposed modifications to the main code branch. Consequently, team members collect comments and suggestions from others who have examined the code and utilize them as valuable inputs when updating it.

Issues provide more than just a mechanism to report defects since they allow others to participate and improve the work's quality. More significantly, the Issue-tracking method helps participants comprehend the project's overall goal and disseminate new ideas for improving the code.

This study aims to answer the following research questions by investigating code review using

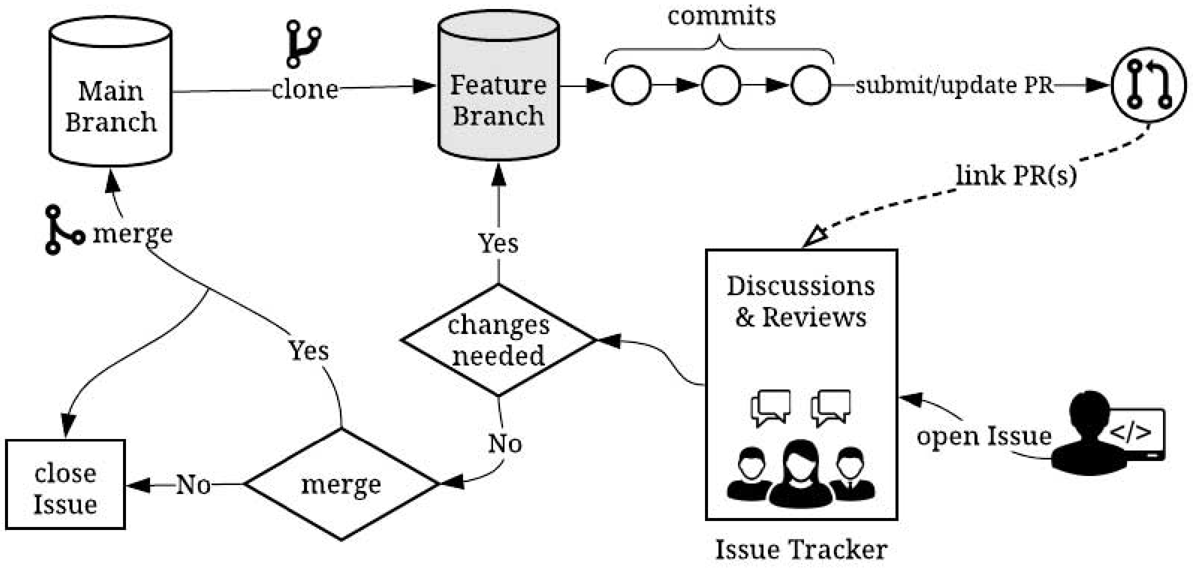


Figure 5.1: Shows the mechanism of Issue Tracker on GitHub.

GitHub Issues:

- **RQ1**: How can we improve the regularity of the code review process?
- **RQ2**: What is the relationship between Issue frequency and community interaction?
- **RQ3:** How involved are the experienced reviewers in Issues?
- **RQ4**: Do more experienced GitHub users receive fewer comments than junior users?

Method:

Dataset

For this project, we collected a large dataset of GitHub repositories and their associated events, including Issues, Comments, and Commits (See Figure 5.2). The dataset is available for download on the *Mega platform* at `https://bit.ly/abdul-dissertation-dataset` under the folder **Chapter05CodeQuality**. We contemplated gathering three sorts of repositories to perform research that would yield accurate results:

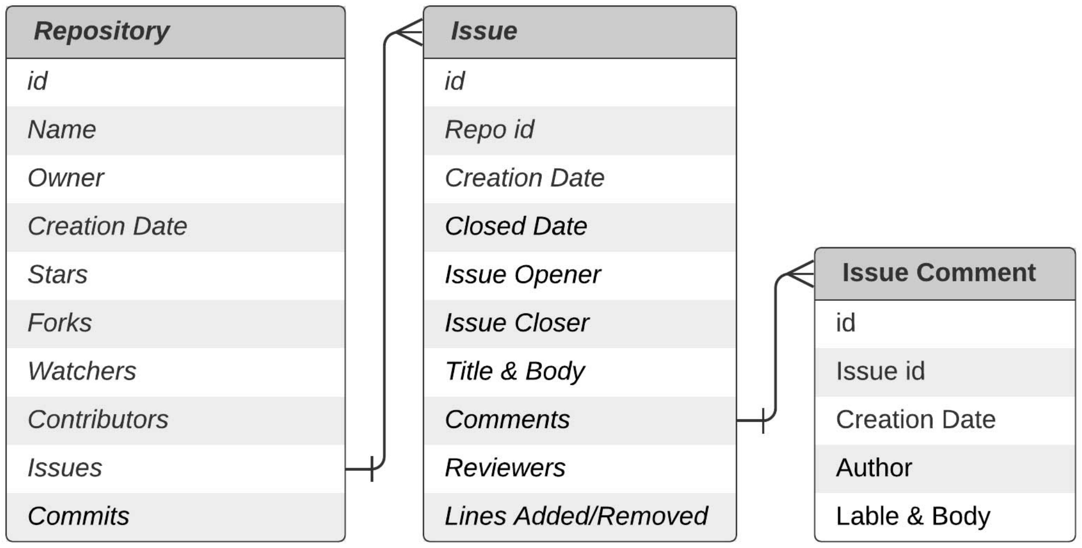


Figure 5.2: Shows a sub-diagram of (Repository-Issue) entities' relation in GitHub. Some of the field names have been modified for simplicity.

1. **Random Repositories**: To obtain such a dataset, we used the GitHub API endpoint */repositories* [70] to retrieve a list of public repositories between March and June 2020. Then, we selected 4,000 random repositories from the list via a Python script utilizing a random number generator function.

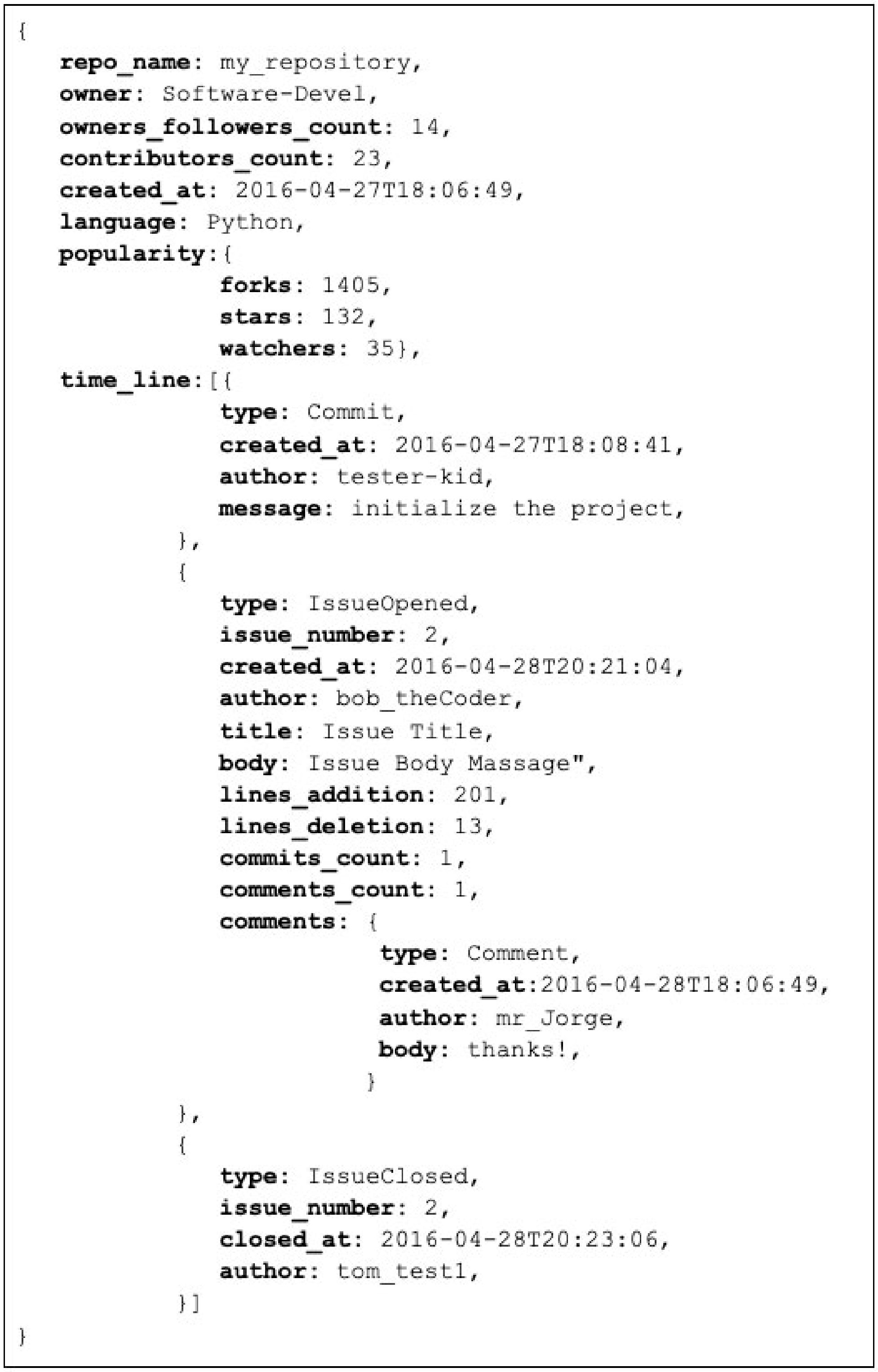
2. **Popular Repositories**: Large projects, organizations, or tech firms operate most of the popular GitHub repositories. On the surface, these teams should adhere to the business criteria that maintain the code quality bar high. As a result, they may be used as a valuable benchmark to compare to other GitHub repositories.

   We first looked at The State of the Octoverse's list of the top five programming languages utilized on GitHub [71] to identify a diverse collection of popular repositories. The top 20 repositories for each language were then picked using GitHub's Trending page, and the data was retrieved using the GitHub API [70].

3. **ROS-Related Repositories**: ROS, or Robot Operating System, is an open-source platform for robotics application development [72]. It provides a stable environment for coding complicated inter-component calculations. When creating robotics-related software, developers must maintain a high degree of quality. This led us to investigate GitHub repositories of this type and create a new benchmark to assess code quality. *GitHub topics* [73] were used to find such repositories. Subject-based labels allow users to browse GitHub repositories by various categories. Using the GitHub API, we retrieved 3,000 public repositories with the term *"ROS"* in at least one of their topics. Table 5.1 details the top 10 topics associated with ROS-related repositories.

Data Preparation: The data for all activities from the day each repository was created was included in the downloaded repositories. As a result, we used a filtration procedure to clean up the downloaded datasets and preserve only the data we needed for our research. In this procedure, we kept all the necessary information about Issues, Commits, Comments, and Pull Requests. Figure 5.3 displays an example JSON data object containing only the necessary data for each repository.

Table 5.2 shows our three repository categories' statistics. Figure 5.4 also shows the distribution of four distinct attributes of the collected repositories. Compared to the other two repo categories,



```
{
   repo_name: my_repository,
   owner: Software-Devel,
   owners_followers_count: 14,
   contributors_count: 23,
   created_at: 2016-04-27T18:06:49,
   language: Python,
   popularity:{
               forks: 1405,
               stars: 132,
               watchers: 35},
   time_line:[{
               type: Commit,
               created_at: 2016-04-27T18:08:41,
               author: tester-kid,
               message: initialize the project,
            },
            {
               type: IssueOpened,
               issue_number: 2,
               created_at: 2016-04-28T20:21:04,
               author: bob_theCoder,
               title: Issue Title,
               body: Issue Body Massage",
               lines_addition: 201,
               lines_deletion: 13,
               commits_count: 1,
               comments_count: 1,
               comments: {
                           type: Comment,
                           created_at:2016-04-28T18:06:49,
                           author: mr_Jorge,
                           body: thanks!,
                          }
            },
            {
               type: IssueClosed,
               issue_number: 2,
               closed_at: 2016-04-28T20:23:06,
               author: tom_test1,
            }]
}
```

Figure 5.3: Presents a sample JSON object representing the needed information for each repository extracted from the collected dataset.

Table 5.1: Top 10 topics labeled on ROS-related repositories.

Topic	Count
robotics	951
deep-learning	920
reinforcement-learning	836
machine-learning	544
deep-reinforcement-learning	418
tensorflow	351
pytorch	315
computer-vision	222
artificial-intelligence	172
robot	166

the popular repositories have more Issues open, as seen in Table 5.2. This is unsurprising and may be linked to large projects or developer teams having more *contributors* than smaller projects or teams.

Although large teams generate more significant numbers of Issues, the code review process is still completed by a small number of *reviewers*: the contributors who review and comment on code through Issues. On ROS-related and Regular repositories, the average number of reviewers is slightly lower than that of large teams on Popular repositories. This might explain why, on average, all types of gathered repositories have approximately the same amount of *comments* per issue.

The *Sentiment Score* of an Issue comment indicates whether the comment is likely positive or negative. This number ranges from -1 to 1, with the greater the sentiment score, the more positive the comment's context. To get this number, we used a Sentiment Analysis tool on Python that runs on a collection of text-processing methods [74].

Given that developers submit code updates through *commits* before each issue, it is worth noting that the number of *added* and *removed* lines of code in Random repositories is slightly greater than

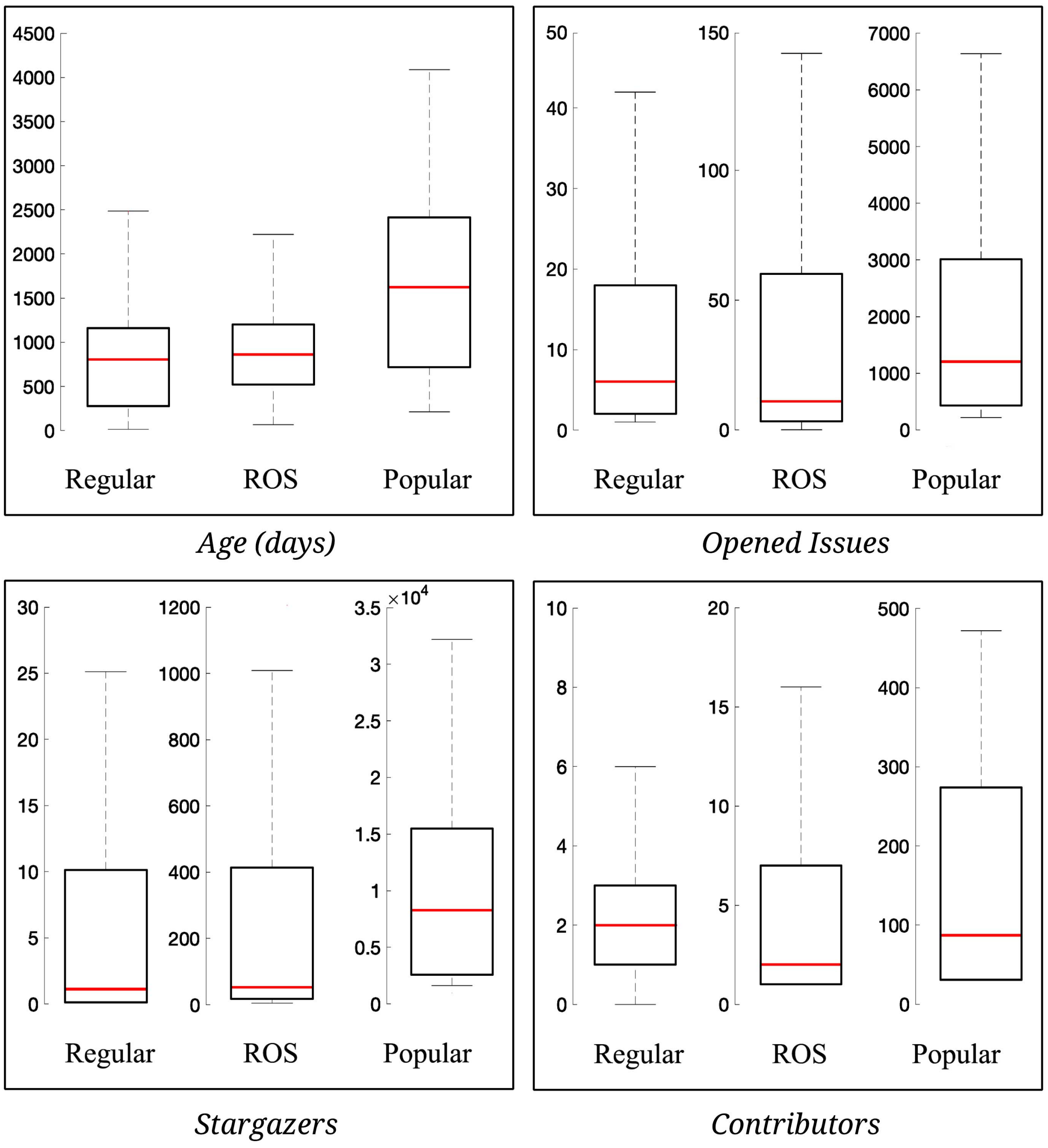


Figure 5.4: Depicts the distribution of the following attributes across all three types of repositories: Age, Opened Issues, Stargazers count, and Contributors count.

Table 5.2: General statistics of the collected repositories.

General Statistics (Avg)	Random Repos	ROS Repos	Popular Repos
Issues per repository	40	130	2155
Closed Issues	89%	88%	89%
Comments per closed Issue	1.19	1.68	3.15
Comments per non-closed Issue	1.36	1.95	3.42
Comments Sentiment Score $[-1, 1]$	0.073	0.085	0.083
Reviewers per Issues	1.31	1.42	2.24
Repository contributors	5.5	10	146
Repository owner followers	81	211	771
Issue opener followers	51	82	213
Issue closer followers	89	185	785
Added / Removed lines per Issue	1690 / 670	1300/650	1330/432
Days prior to the first Issue	128	102	71
Hours to open an Issue	450.94	170.5	25.2
Hours to close an Issue	13.88	15.61	15.38
Commits per repository	216	695	10776
Commits prior the initial Issue	83	102	753
Commits between Issues	5.42	6.35	8.48

in ROS-related or Popular ones. To put it another way, developers on Regular repositories are more likely to make a large number of code changes before committing them to the repository. When it comes to code review, however, Issues on Regular repositories receive fewer comments than other types of repositories. This finding might indicate lower-quality work on Regular repositories when comparing the number of new codes to the number of comments through open Issues.

Developer activity is an excellent place to start examining Issues on GitHub repositories. As shown in Table 5.3, teams with more contributors are more likely to have more Issues than repositories with fewer contributors. Popular vs. less popular repositories exhibit a similar pattern: repositories with more Stargazers, Watchers, or Forkees are more likely to have more Issues opened. Figure 5.5 depicts the correlation between open Issues to contributions and stargazers.

Table 5.3: The Correlation between the total Issues counts to other features for all three different types of the collected repositories.

	Regular		**ROS**		**Popular**	
	R-Val	*P-Val*	*R-Val*	*P-Val*	*R-Val*	*P-Val*
Repo Age	0.217	1.28^{-44}	0.216	1.89^{-33}	0.406	1.54^{-5}
Contributors	0.46	8.75^{-213}	0.812	0	0.765	7.59^{-15}
Issue Comments	0.914	0	0.925	0	0.781	4.93^{-21}
Commits	0.438	8.43^{-191}	0.728	0	0.451	1.21^{-6}
Reviewers	0.194	8.56^{-36}	0.194	3.41^{-27}	0.254	0.008
Commits before the first Issue	0.058	0.00019	0.166	4.15^{-20}	0.139	0.154
Commits between Issues	−0.019	0.216	−0.026	0.142	0.107	0.271
Issue Opener Followers Count	0.053	0.0006	0.045	0.129	0.142	0.146
Repo Owner Followers Count	0.077	6.9^{-7}	−0.013	0.446	−0.246	0.010
Stars	0.387	5.76^{-146}	0.457	7.79^{-156}	0.311	0.001
Forks	0.351	1.19^{-118}	0.409	1.43^{-122}	0.452	1.13^{-6}
Watchers	0.391	6.73^{-149}	0.404	4.18^{-119}	0.328	5.7^{-4}

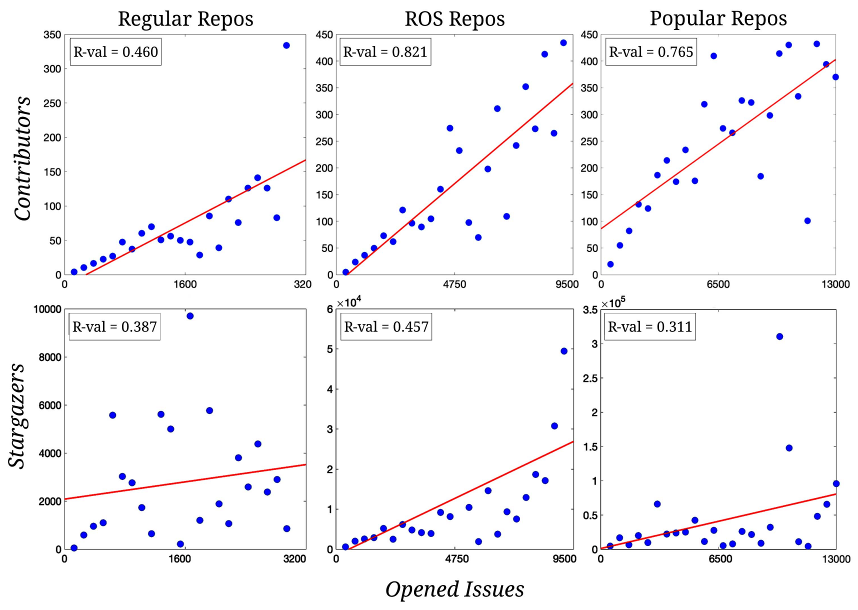


Figure 5.5: Top row: Correlation between opened issues and total contributors. Bottom row: Correlation between opened issues and total repository stars.

In addition, the number of total reviewers on a repository is related to the number of Issues. Opening more Issues may increase Contributor participation, suggesting improved collaboration and teamwork.

Code Review Process via Issue Tracking mechanism

The issue-tracking mechanism enables team members to interact, share information, and review code. A code review is one of the most effective ways to enhance code quality. Furthermore, code review aids in the mentoring of engineers with less expertise. It also improves teamwork, fosters a collaborative mindset, saves project time or cost, and distributes information.

Consider a situation where two developers work in a Repository *R* on implementing a single software feature. Developer *A* is responsible for the back-end implementation, whereas developer *B* is responsible for the user interface and front-end development. They clone the main repository to their local branches before getting started. They propose merging their new code from the local branches to the main branch via separate pull requests after completing the implementation.

One of the team members opens an Issue to discuss the new feature that the developers have developed, allowing everyone to express their ideas and opinions. The Issue opener links both submitted pull requests to the Issue and makes them available for other contributors to code review. Contributors can express their opinions on the proposed code and how developers *A* and *B* implemented the feature by leaving comments. A collaborator can also reference other users by tagging them in a comment inside the opened Issue to be notified. A collaborator can tag additional team members in a comment, providing a flexible approach to get the relevant people involved in the code review.

Developers *A* and *B* read the comments and participate in the discussions to better understand what other users think and suggest. If an Issue is discovered, code reviewers may provide recommendations or adjust the developers' code. Developers *A* and *B* review the remarks and make the necessary code changes. After all these interactions and discussions between the developers and code reviewers, a contributor merges the pull requests into the main repository and closes the Issue once everyone is pleased and comfortable with the suggested adjustments.

In this scenario, the team discussed an implementation using Issue Tracker (See Figure 5.1). This process contributes toward improving the overall quality. Moreover, roughly 90% of all open Issues are closed across all three repository types in our dataset. On the other hand, the Popular repositories have a greater rate of comments per Issue (See Table 5.2). This means that popular repositories place a greater priority on debating suggested code changes and exchanging ideas before accepting a new update into their main repository (See Figure 5.6).

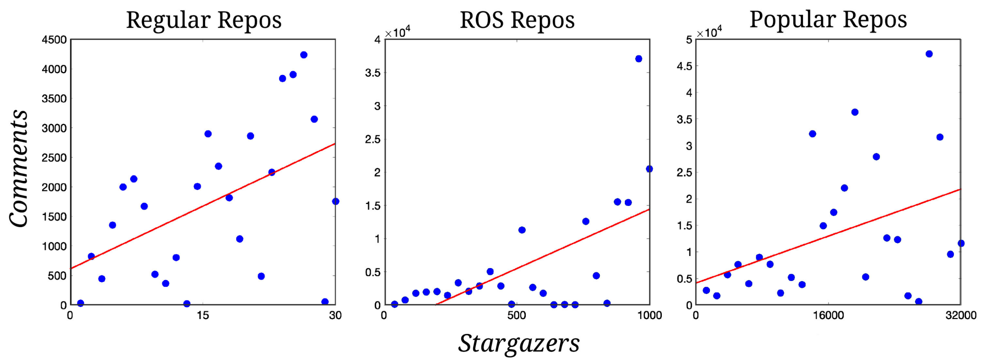


Figure 5.6: Presents the correlation between the number of repository stargazers and the total number of comments received on Issue.

According to Table 5.2, it takes roughly six months for developers on Regular repositories to open their first Issue. This number is around 5 and 2 months in ROS-related and Popular repositories, respectively. On average, teams on Popular repositories open a new Issue every 25 hours. Teams on the ROS and Regular repositories appear to take longer to open new Issues. However, once contributors open an Issue, the teams, regardless of the repository types, will examine and close the Issue in roughly the same amount of time. Most Issues are opened and closed within a few hours;

however, a small number remain open for an extended time. There might be various causes for having non-closed Issues. However, the average number of comments on such Issues is somewhat greater than on other Issues, as shown in Table 5.2, which could suggest that the reviewers were still not satisfied with the code's quality.

Collaborators and Reviewers

The concept of cooperation is one of the aspects that makes Issues a robust tool for achieving code quality. When an Issue is opened, contributors will be notified and can engage in the conversation and give their opinions. The more popular a repository becomes, the more contributors it has, as shown in Table 5.2. More contributors indicate a more extensive scope of work that requires a large team to complete, resulting in more code being generated. As a result, contributors will submit more pull requests, and hence, a more significant number of Issues may be opened in repositories with higher popularity. More Issues has a solid connection to code review; thus, contributors will provide more comments (See Table 5.3). On the other hand, more contributions to a repository do not always indicate a higher number of reviewers, as seen in Table 5.2, since our dataset did not demonstrate such a strong connection. Regardless, the number of reviewers that perform code reviews and post comments is relatively similar across all types of repositories in our sample. The contributors' popularity (followers count) directly involved in opening, closing, and evaluating Issues is detailed in Table 5.4. The data in this table demonstrate that users who open Issues are less popular than users who close Issues or review code. Assuming users' popularity is proportional to their expertise, the submitted codes should satisfy highly experienced engineers before code merging. Low-quality code will not be merged into the repository until senior contributors have reviewed it. This process enables the team to modify the code and adhere to high-quality requirements before merging it.

Table 5.4: The followers count for users open/close Issues and Issue reviewers.

	Random Repos	ROS Repos	Popular Repos
Issue opener followers	51	82	213
Issue closer followers	89	185	785
Reviewers followers	71	180	542

Issues Density

Examining the time intervals between GitHub Issue opening events might give a more accurate picture of how a team's activities affect code quality. Issues on GitHub can be created at any moment by any contributor. When we look at the life cycle of a repository, we can observe a succession of Issues that are opened at different periods. A time *distance* (d) is the time repository contributors spend between two adjacent Issue opening events.

$$d_i = t_{I_{k+1}} - t_{I_k} \tag{5.1}$$

t_{I_k} denotes the time t at which the Issue I of order k is opened, where $Issues_R = \{I_1, I_2, .., I_k, I_{k+1}, .., I_n\}$ and n = the total number of the Issues in repository R. The set of distances between Issue events of repository R will be as follows:

$$D(R) = \{d_1, d_2, ..., d_{n-1}\} \tag{5.2}$$

where we have $n-1$ pairs of adjacent Issues for a set of n Issues. We can also see that some Issues are opened in a short period of time, with relatively short time distances between them, while other Issues have a longer time distance between them. Figure 5.7 depicts a subset of Issues opening events on a repository timeline.

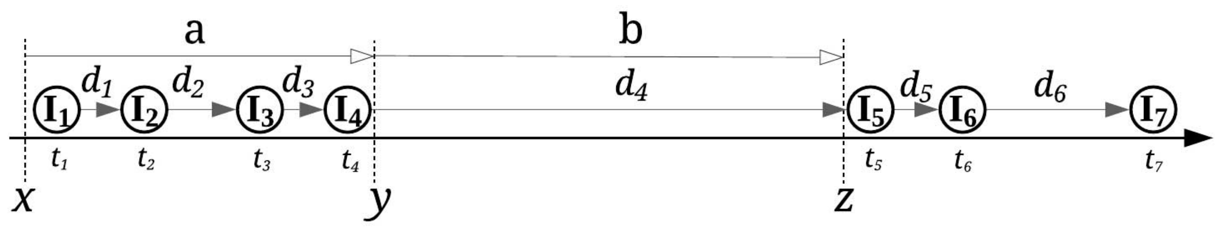


Figure 5.7: A repository Issues openings events. $a = \{I_1, I_2, I_3, I_4\}$ and $b = \{\}$ while $d_a < d_b$.

Issues opened in a short time interval, such as Issues on a time interval (a), are referred to as *dense Issues* in this work. Some Issues are far apart and have a long time interval between them, with no conversations or code review (using the Issue Tracking mechanism), such as the time interval (b) in Figure 5.7. We refer to Issues (like I_4 and I_5) that are separated from one another as *dispersed Issues*.

We see this behavior in most of our dataset's GitHub repositories, where contributors open many Issues in a short period and spend a lengthy time initiating a new Issue during other phases of the software development cycle. The distribution of Issue distances set $D(R)$ of repository R may be used to determine how dense or dispersed Issue events are. Figure 5.8 depicts such a distribution for one of the repositories in our dataset.

The more dense Issues we have, the more left-skewed the distribution becomes; conversely, if there are more dispersed Issues, the distribution becomes right-skewed. Because the long tail of a skewed distribution lies in either the left or right direction of the number line, the *median* better represents the data's center location than the *mean* value.

The temporal distances between any pairs of Issues opened within a time span close to the median value are represented by the data on or around the median. These are referred to as *Regular Issues*. In this study, the Regular Issues were considered to be within one standard deviation of the

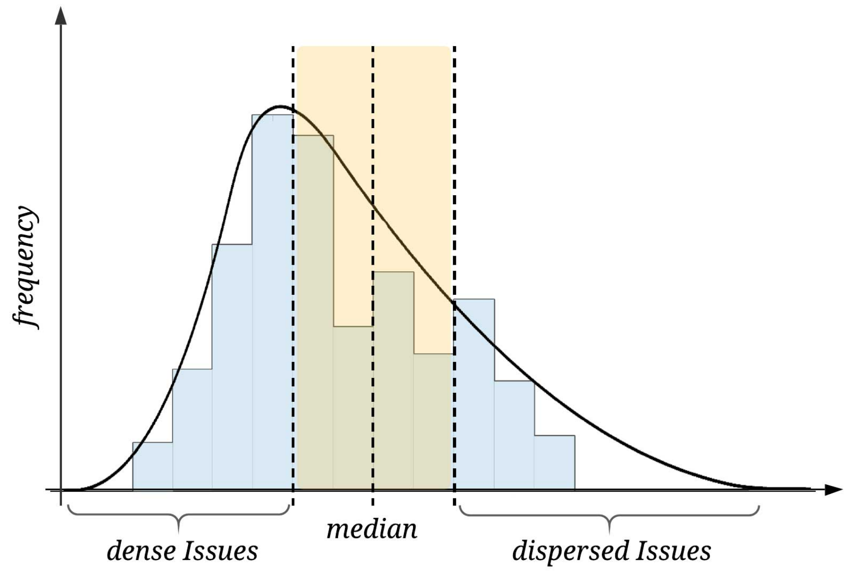


Figure 5.8: Displays a left-skewed distribution of the time intervals between open Issues.

median, $[median \pm 1std]$. As a result, the range below $(median - 1std)$ reflects Issues that were opened relatively quickly (dense Issues), whereas the range above $(median + 1std)$ represents the distribution of dispersed Issues distances. Table 5.5 shows the distribution of dense, regular, and dispersed Issues for the repositories in our dataset.

Table 5.5: The distribution of Issues opening time.

	Dense Issues	Regular Issues	Dispersed Issues
Popular Repos	9.06%	72.73%	18.21%
ROS Repos	15.70%	62.09%	22.21%
Random Repos	10.96%	67.49%	21.55%

Issues Opening Regularity

Rather than examining the total number of Issues, we hypothesize that examining the time intervals between GitHub Issue opening events might provide a more accurate picture of how a team's activities affect code quality. We have divided the Issues into three categories: dense, regular, and dispersed, based on how close they are to the median of the time distance distribution.

Dense issues reflect rapid team activity through Issue Trackers. Contributors participate in activities and conversations to quickly examine and evaluate new code through various Issue events. Various factors might cause many issues to arise within a short period. This is, nevertheless, normal team behavior, particularly when a significant new feature is merged into the main branch via a series of pull requests. As a result, the team must devote extra time to evaluate the code quality before it can be accepted.

On the other hand, having numerous consecutive Issue events separated from one other suggests that there are fewer social interactions amongst the contributors of a GitHub repository over time. In such cases, contributors may still need to complete a code implementation to evaluate or discuss using Issue Trackers when a team conducts multiple code merges outside the Issue Tracking system. In contrast, the rate of dispersed Issues is high, which can result in a substantial gap in knowledge transfer, technical conversations, and thought exchange, all of which can influence the team's overall code quality.

On the other hand, issues in the regular distance range suggest continuous teams' code review behavior, in which contributors frequently interact through participation in social activities through Issue Trackers. We refer to this team behavior as ***Regular Code Reviews***. Repositories with a greater rate of Regular Issues have a higher chance of performing regular quality checks, which helps to maintain and improve the code's high quality.

According to [75], code review has been ranked as the most effective way to achieve better quality. Several annual studies on software quality looked at code reviews and attempted to determine which elements had the most impact on quality improvement. According to [75], regular code review has been the most effective approach for improving quality; the following section describes our proposed mechanism for promoting regular code review.

Results

In this section, We provide the findings of our investigation into the Issue Tracking system and the code review process on GitHub with respect to our four research questions.

Issue Regularity

RQ1: How can we improve the regularity of the code review process?

The code quality of GitHub projects is affected by several variables. As previously discussed, one of the most critical approaches to enhance quality is to perform frequent code reviews, which may be conducted through the GitHub Issues Tracking system. To do this, we study the repositories' timeline and investigate the distribution of temporal distances between Issues to create a reminder mechanism to attain *Regularity*. We propose the *New Issue Notifier* (*NIN*), a new mechanism that assesses contributors' behavior over time (Figure 5.9).

To determine the timing of prompts, NIN leverages the median of the Issues' temporal distance distribution. The suggested time to open an Issue is not static, as we continually recalculate it based on the median over time. We use this method to decide when to notify the team, reminding them that it may be time to open an Issue, which they may either accept or decline.

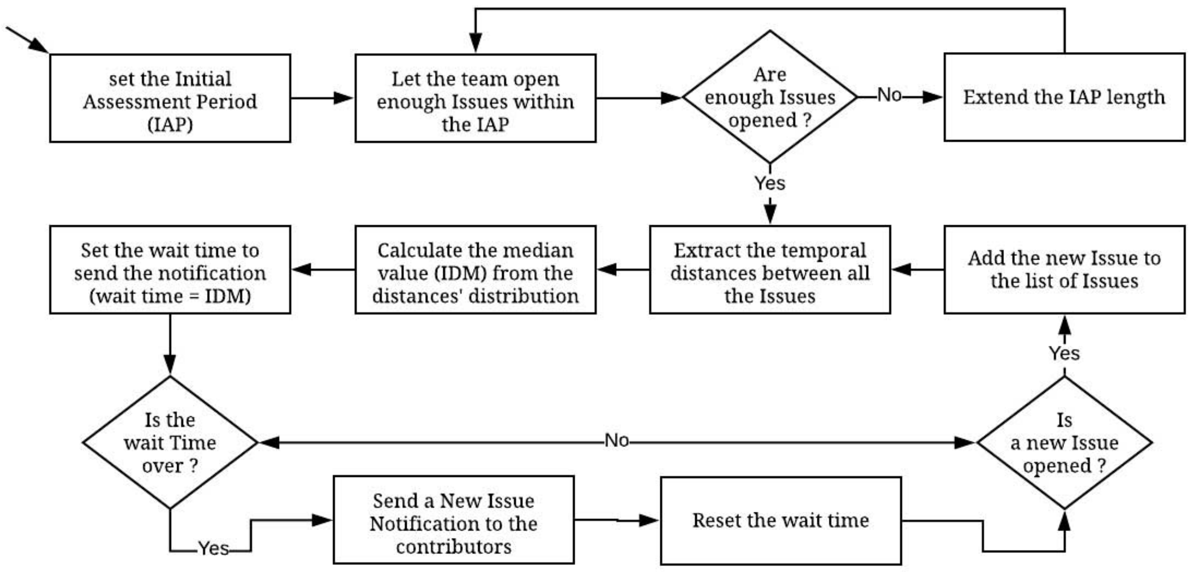

set the Initial Assessment Period (IAP)
Let the team open enough Issues within the IAP
Are enough Issues opened ?
No
Extend the IAP length
Yes
Add the new Issue to the list of Issues
Extract the temporal distances between all the Issues
Calculate the median value (IDM) from the distances' distribution
Set the wait time to send the notification (wait time = IDM)
Is the wait Time over ?
No
Is a new Issue opened ?
Yes
Yes
Send a New Issue Notification to the contributors
Reset the wait time

Figure 5.9: Shows the New Issues Notifier (NIN) mechanism.

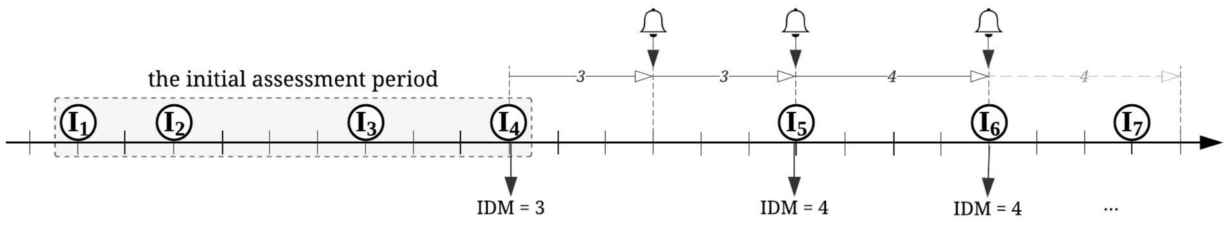


Figure 5.10: Presents the New Issue Notifier mechanism (NIN) over a GitHub repository timeline. The *IDM* stands for Issue Distances Median and recalculates each time a new Issue is opened.

The process starts by retrieving the history line of the repositories' Issue Opening events, as illustrated in Figure 5.10. To initialize the system, we need enough Issue-related data to evaluate the teams' behavior regarding opening Issues. The *initial assessment period* is defined as the time frame during which we monitor the teams' activities in terms of opening new Issues for this purpose. The initial assessment time would be extended if we could not collect enough data.

The temporal distances between the Issues opened during the assessment time frame are extracted at the end of this period. (In Figure 5.10 the temporal distance set $D = \{2,4,3\}$ considering that the assessment period included the Issues $\{I_1, I_2 I_3, I_4\}$). Afterward, we calculate the Issues Distances Median (*IDM*), the average time contributors spend opening new Issues throughout the development's life cycle.

To determine the best time for the team to open a new Issue, we utilize the *IDM*, extracted from the team's activity pattern. In NIN's mechanism, the *IDM* is the waiting time to push a notification to the contributors if the team has not already opened an Issue. The notification suggests opening a new Issue; however, the team can act upon the notification and start a new Issue or ignore it and continue without opening any Issue. The notification will be scheduled to trigger repeatedly after the waiting time is over. If the team opens any new Issue (before or at the notification time), the *IDM* is calculated again, but this time for a more extensive set of Issues, including the new one. The more Issue data we have, the more accurate the *IDM* becomes. Algorithm 3 details the steps of our proposed approach.

We developed a simulation of the New Issues Notifier, or *NIN*, considering the following key factors:

- The examination included all repositories from all three repository categories.
- All repositories with an insufficient number of Issues were deemed noise data and were thus eliminated before the simulator was run.
- The simulator was fed a collection of actual ground truth Issue data from each repository for the initial assessment period as its first input.

- Table 5.2 shows that contributors spend an average of 2 to 4 months to open the first Issue across all three types of repositories. After that, the pace of opening Issues rises, and they tend to open Issues several times monthly. As a result, the initial assessment period (IAP) has been set at six months from the day the repository was created to allow us to collect adequate data.

- We defined three *Acceptance Probabilities* (AP): 0.3, 0.6, and 0.9. They reflect the likelihood that the team will accept the New Issue notification.

The simulator was used to recreate three years of GitHub team activity. The simulator operates on three threads, each with its acceptance probability (IAP). Once the execution was completed, we retrieved our repositories' Dense, Regular, and Dispersed Issues distribution. In general, as indicated in Table 5.6, we can see an increase in the number of issues opened more frequently compared to the data in Table 5.5.

Table 5.6: The percentage of dense, regular, and dispersed Issues after running the simulator for each acceptance probability (AP).

Repositories Type	AP	Dense	Regular	Dispersed
Popular	0.3	9.60%	79.63%	10.77%
	0.6	9.78%	81.13%	9.09%
	0.9	10.52%	81.10%	8.38%
ROS	0.3	16.15%	66.54%	17.31%
	0.6	14.26%	70.90%	14.83%
	0.9	13.39%	74.47%	12.14%
Random	0.3	12.12%	69.45%	18.43%
	0.6	10.89%	72.25%	16.86%
	0.9	9.99%	74.95%	15.06%

A higher acceptance ratio increases the number of Regular Issues. Compared to ROS and Random repositories, this change is slightly less in Popular repositories. Figure 5.11 shows a collection of heat maps depicting the percentage of Issues that are opened more regularly before and after the simulator was performed. The figure indicates that the Regular Issues rate rises over time as more Issues are opened, irrespective of the chance of notification acceptance.

The New Issue Notifier method (NIN) is an approach that uses the Issue Tracking system to improve regularity. It nudges the team to collaborate more frequently to discuss the new implementation and review the submitted code before merging, which may improve code quality over time.

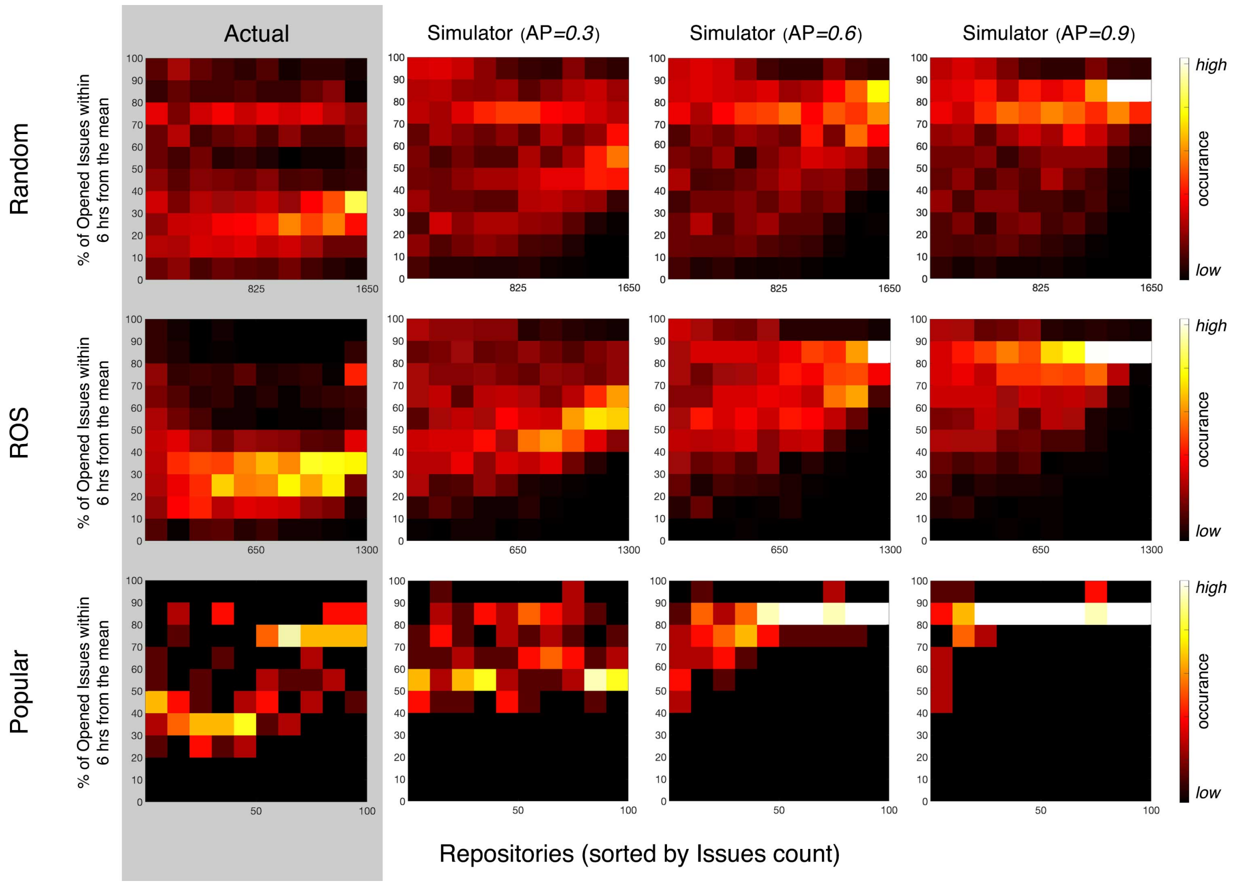


Figure 5.11: Shows the percentage of the Issues that fall within the regular area $[median - \alpha, median + \alpha]$, where $\alpha = 6$ hours, for all three types of repositories before and after running the simulator (for three different acceptance probabilities).

Issues Community

RQ2: What is the relationship between Issue frequency and community interaction?

Contributors open Issues on GitHub to allow the team to collaborate and exchange information. Issues may be seen as parts of a larger community within a repository from a broad perspective. As a result, when it comes to quality, we should consider the larger community, where users contribute to a greater goal: a *high-quality product*. We built a network of Issues and their reviewers to comprehend users' engagement in repositories' Issues and explore their interactions. This network is multi-layer, with node connections on the first layer defining connectivity on the second layer. From the standpoint of code review, this network reflects the Issues Community of GitHub repositories. The network is formed as follows:

- Node type 1 (n_1): represents repository R's opened Issues $\{I_1, I_2, ..., I_n\}$, where n is the total number of the Issues.
- Node type 2 (n_2): represents the users $\{r_1, r_2, ..., r_m\}$ who reviewed the proposed code via Issues in repository R, where m is the total number of the reviewers.
- Edge type 1 (e_1): a weighted link that connects r_x (n_2 nodes) to I_y (n_1 nodes), if r_x has reviewed the code through I_y, where $1 \leq x \leq m$, and $1 \leq y \leq n$. The more comments r_x has through I_y, the more weight the edge (r_x - I_y) gets.
- Edge type 2 (e_2): a weighted linked that connects I_i to I_j (n_1 nodes) if r_x has reviewed codes through both Issues I_i and I_j.

Figure 5.12 shows a sample Issue Community of repository R, with reviewers nodes $r1$, $r3$, and $r4$ connected to several open Issues. Compared to others, these nodes indicate contributors can provide help across Issues. These reviewers are more likely to participate in conversations, share their opinions, and, in general, help their teammates evaluate their work and merge any code that meets high-quality standards. Figure 5.13 depicts three separate Issue Communities and their scores. The graphs only show edges of type E_1 to make it more visually apparent.

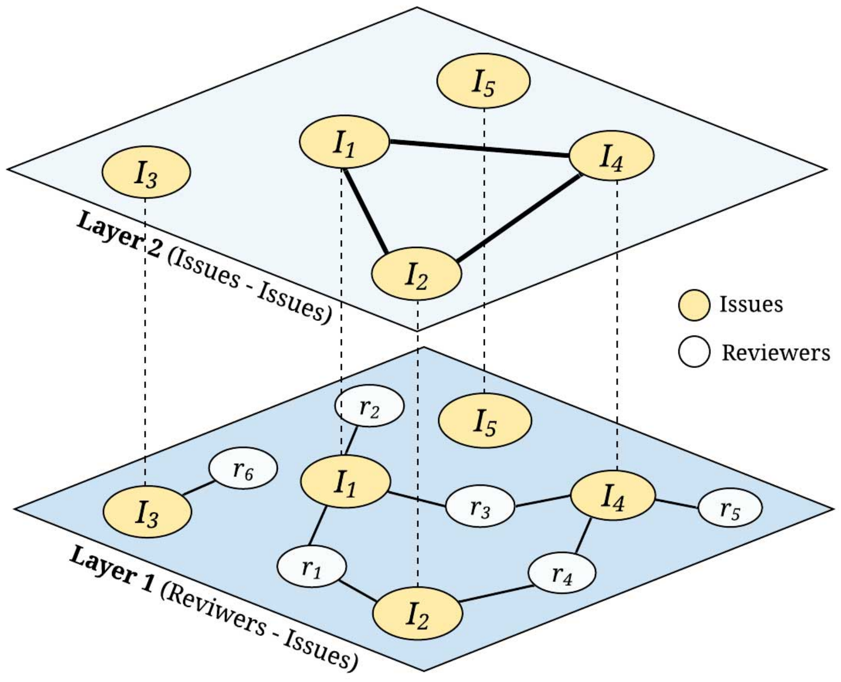
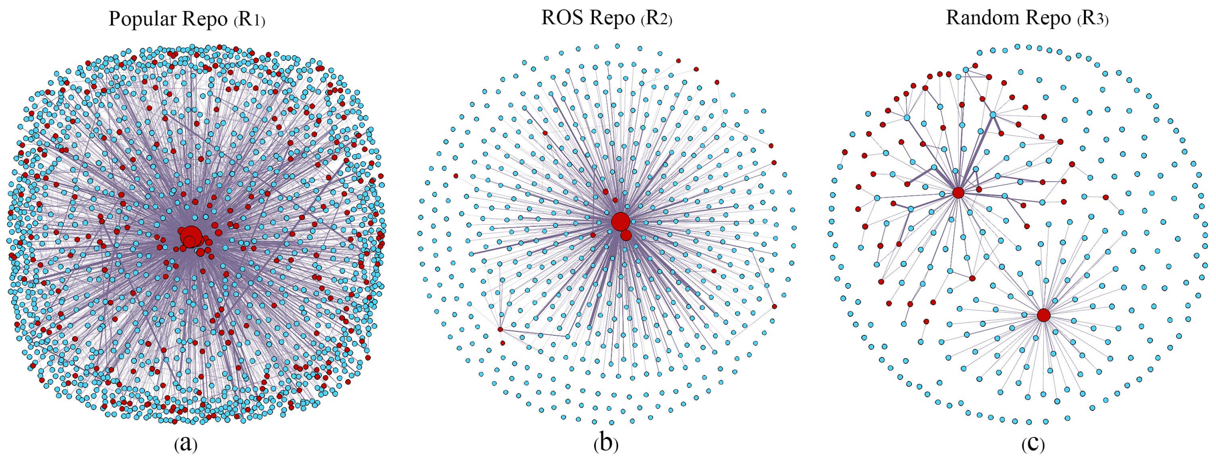


Figure 5.12: A sample Issue Community (*IC*) network is formed by two types of nodes that represent Issues and reviewers. There are two layers: Layer 1 shows the edges of type e_1 that connect Reviewers to Issues, and Layer 2 shows the edges of type e_2 that connect Issue to Issue.

Reviewer collaboration behaviors vary depending on the repository and the project they are working on. Some repositories have proactive and experienced team members willing to participate in technical reviews and conversations for the team's benefit, while others have less active users. As a result, we defined the *Issue Community Score* (*ICS*) to assess each repository's Issues Community in terms of user involvement. The *ICS* represents the flow of information, idea sharing, thought



Graphs	ICS	Nodes	Edges	Issues	Comments
(a) Popular Repo (R_1)	0.819	1502	2122	1241	3727
(b) ROS Repo (R_2)	0.763	553	967	537	1687
(c) Random Repo (R_3)	0.478	303	192	252	304

Figure 5.13: Displays sub-graphs of three different Issue Communities, with just the e_1 edges shown (reviewers-Issues connections). The red nodes represent the reviewers, while the blue nodes represent the open Issues. The degree of a node is determined by its size.

interchange, and the interconnectedness between Issues and Reviewers. We have calculated the ICS for each repository as follows:

$$ICS_R = \frac{|IC_R(e_2)|}{|Issues_R - 1|} \tag{5.3}$$

The numerator is the total number of the edges of type (e_2) within the Issue Community of the Repository R. The score is not guaranteed to scale to 1.0, however, having more edges connecting Issue nodes leads to having a higher score which indicates how much contributors' expertise and knowledge are shared through Issues. On the other hand, a lower ICS shows that there are distinct Issues that are not connected to other parts of the community. It is also possible for the ICS score to be lower and the network not to be sparse, but there are only a few edges linking the Issues.

Figure 5.12 also shows an example in which I_3 and I_5 are separated from other Issues. I_5 is reviewed by r_6 who is not involved in other issues, while I_5 is opened and closed with no reviews.

Table 5.7 shows the correlation between *ICS* and other features. Compared to other characteristics such as repository contributors count, stargazers, or the total number of Issues, we found that *ICS* strongly correlates with the total number of Issues comments. This correlation appears stronger in popular repositories, suggesting repository contributors use Issues to communicate, share ideas, and review codes. (See Table 5.2).

Table 5.7: The Correlation of repositories properties: Issues, Contributors, Stars, Issue Comments and the Issue community score.

	Random		**ROS**		**Popular**	
	R-Val	*P-Val*	*R-Val*	*P-Val*	*R-Val*	*P-Val*
Contributors	0.146	0.061	0.226	0.001	0.296	2.788^{-3}
Stargazers	0.124	0.113	0.235	7.703^{-4}	0.167	0.095
Issues Count	0.219	0.004	0.115	0.102	0.227	0.022
Comments	0.359	2.33^{-7}	0.259	1.966^{-4}	0.427	9.17^{-6}

Teams who open Issues regularly may only sometimes have higher code quality. From the standpoint of Issue Tracking, greater code quality necessitates the opening of more Regular Issues and the participation of more contributors through Issues and collaboration as a single team. Investigating our dataset, we have noticed that repositories have many of their Issues opened and closed with only a couple or no reviews. Hence, user participation, involvement, and communication are as crucial as Issues Regularity.

Expertise Coverage

RQ3: How involved are the experienced reviewers in Issues? When code is submitted through an Issue to be evaluated and discussed by other contributors, it is critical that the highly experienced team members participate in the process. With their seniority, expertise, ideas, and general high-level vision, experienced contributors assist the team. The quality of the work will be improved if more of these users contribute directly to a larger number of Issues. A repository with most of its Issues reviewed by experienced individuals will likely have better code quality than one with most of its Issues reviewed by less experienced team members. Based on this assumption, we studied the Expertise Issue Coverage on different repositories.

When developers register a GitHub account and begin their coding adventure, their publicly available coding material usually attracts other users. In GitHub, users can follow other developers and receive notifications when the followee publishes new content; users are likely to be attracted to more experienced developers who produce higher-quality work. As GitHub users maintain a high level of competence, their popularity increases over time, and they gain more experience. As a result, the number of people who follow a user is proportional to their seniority and experience level. The individuals who have the most followers are likely the most experienced. (See Table 5.4).

The Issue coverage by reviewers is depicted in Figure 5.14, with reviewers ordered from highly experienced to low experienced. The graph demonstrates that all three types of repositories follow the same overall expertise coverage pattern, with the most popular and experienced users representing roughly 20% of the total reviewers.

Table 5.8 shows that, on average, 90% of repository users' followers follow the most popular 20% of the reviewers. Regardless of repository type, the 20% most popular reviewers cover around 60% of all Issues within their repositories to assess other team members' work and provide feedback

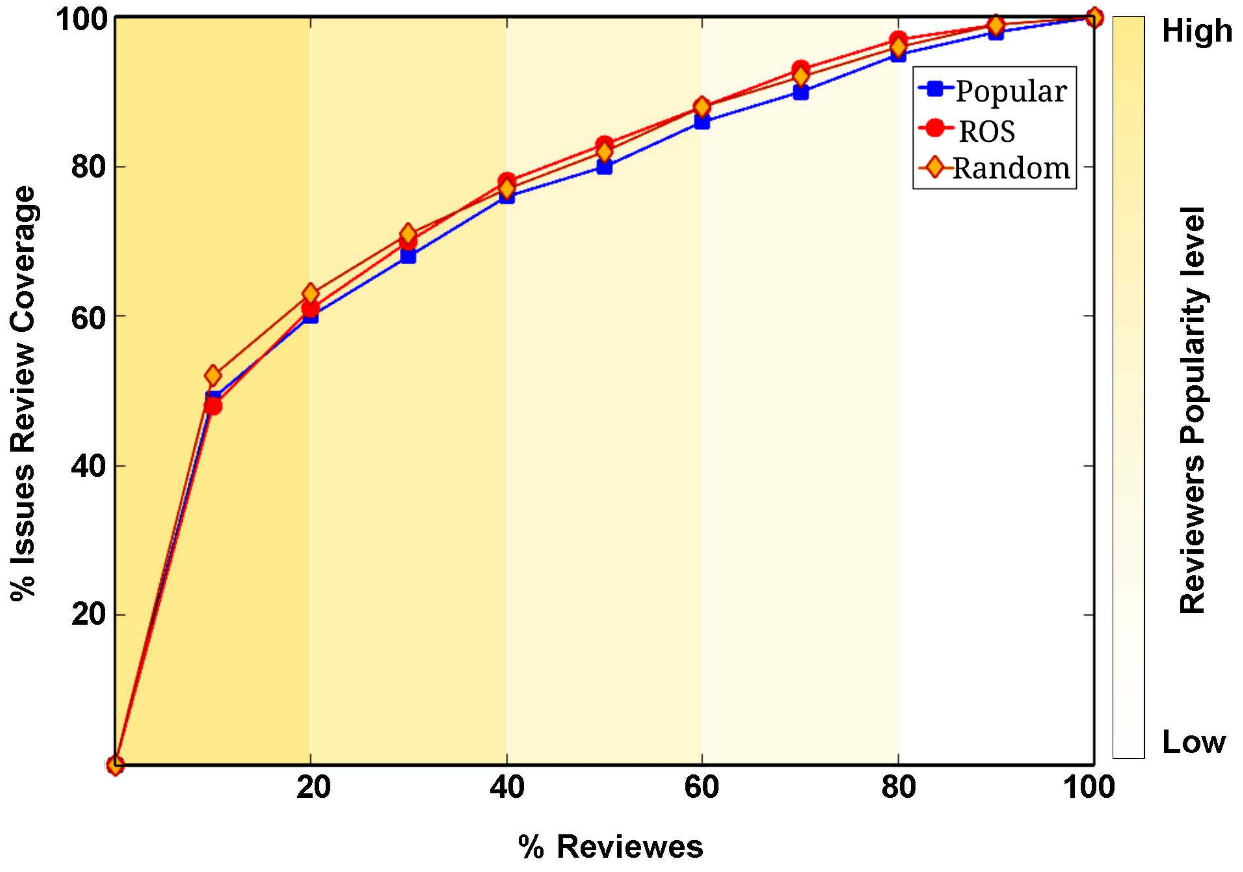


Figure 5.14: Shows the Issues coverage by Reviewers for all three types of repositories. The reviewers are sorted from high to low by popularity (follower count).

Table 5.8: Shows the popularity ratio and percentage of the Issues that 20% of most popular reviewers cover. On average, 20% of the reviewers (the most popular ones) cover more than half of the Issues.

Repositories Type	Popularity Ratio	Issue Coverage
Popular	93%	59%
ROS	90%	60%
Regular	87%	61%

and correction. Although the data indicate a similar Issue coverage pattern for Popular, ROS, and Regular repositories, the levels of popularity and expertise vary. As shown in Table 5.4 on Popular repositories, contributors closing an Issue are far more prevalent and thus experienced than on ROS and Random repositories. As a result, regardless of the similar coverage pattern, the quality of the review and assistance on the Popular repositories might be significantly greater.

Nonetheless, the expertise coverage percentage shows that when it comes to code reviews, popular reviewers play an essential role in spreading their opinions, ideas, and experience through Issues, which assists other less experienced developers in learning how to improve their work quality.

Developers Popularity

RQ4: Do more experienced GitHub users receive fewer comments? When a contributor opens an Issue and links a pull request to it, others can participate in reviewing, sharing their thoughts, and suggesting updating the code if a revision is needed. Now imagine if a perfect high-quality code is linked to an Issue for review. This code may receive fewer reviewer comments than another code that must be updated. Many factors can impact the quality of the code, but we assume that the seniority and the developer's experience can significantly impact his/her code's quality.

To address research question 4, we examine the relationship between Issues openers' popularity and the comments they receive through Issues, considering that a user's popularity is determined by the number of followers they have. Only Issues in which their openers had submitted their code for review were included in this investigation.

Figure 5.15 demonstrates the relationship between the popularity of Issue openers and the number of comments an Issue receives, sorted by Issue number. Each data point on the graph represents data from a single repository; the higher the data point, the more comments the popular Issue

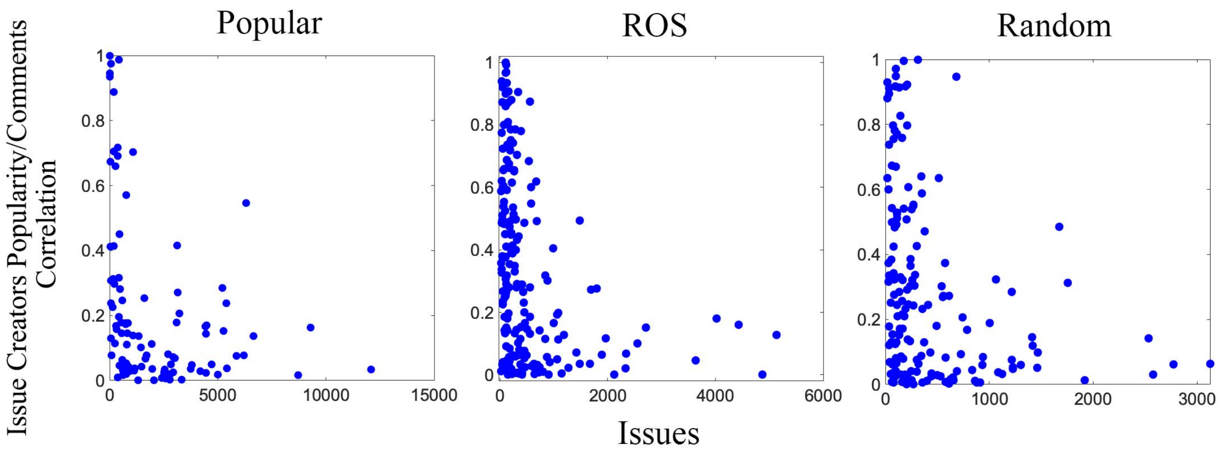


Figure 5.15: Illustrates the correlation between Issues openers' popularity and the comments they receive, sorted by Issue count. The popularity of a user is its total number of followers on GitHub.

openers will receive.

There is a considerable correlation between Issue openers' popularity and the number of comments they receive, as seen in Figure 5.15 for repositories with fewer Issues. This means that even if a team member who opens an Issue is popular, he or she will still receive much feedback. We have found this trend in most of the repositories we have examined.

On the other hand, we see a different pattern in the repositories with many Issues. The popular users who open Issues, regardless of repository category, receive fewer comments on such repositories. In such repositories, the less popular users may receive more attention when their code is reviewed, suggesting that others may not initially approve it.

As a result, being a popular developer in most repositories does not always imply receiving fewer corrections or comments or getting their codes reviewed with no absolute comments. This behavior indicates that most developers, including the experienced ones, will have a thorough quality check on their code, resulting in improved code.

Threats to Validity

Our study assumes that using social coding platforms to enforce good software engineering practices such as review regularity, increasing community involvement in code review, and recruiting more senior software engineers to participate in the review process is likely to yield higher quality code; however, following best practices is not a guarantee of ultimate code quality. Moreover, reminder mechanisms such as the New Issue Notifier may overload developers simultaneously involved in multiple projects. Our analysis did not include team communications that occurred via Slack or video conference.

Conclusion

This chapter presents an analysis of the code review process when conducted using the GitHub Issue tracking mechanism. We collected data from three types of GitHub repositories: 1) popular repositories (owned by big tech projects, organizations, or companies), 2) ROS-related repositories (robotic-specific implementations with an active user community), and 3) randomly selected public repositories.

First, an analysis was conducted of how Issue frequency and timing are affected by repository features such as age and the number of contributors. Based on these findings, we propose a mechanism, *New Issues Notifier* (NIN), which nudges developers towards greater regularity in the review process by prompting them to open issues during dormant periods. In this new approach, we set an initial period to collect enough Issue opening patterns. Then, we continuously calculate the following time points based on the most recent set of Issues to notify the team to open a new Issue. This approach was simulated and executed with three threads considering different user acceptance probabilities on each thread; we show that even low user acceptance yields greater regularity during code review.

This chapter also designed a multi-layer network representing the Issue Community, where the users are linked to Issues if they review a code through them. Based on that, we introduced a new metric (Issue Community Score) for evaluating community involvement and collaboration in the code review process through the issue tracking mechanism. Although we believe enforcing regularity is an essential aspect of the code review process, it does not necessarily yield greater community involvement since it is possible for repositories with high regularity to have a low *ICS* score. To gain a more robust Issue Community, contributors need to have a higher interaction rate across a higher number of issues, which may lead to improved communication, knowledge spread and thought exchange.

Fortunately, even with the current GitHub Issue tracker, most repositories successfully recruit senior software engineers to participate in code reviews. Based on our research, offering other recognition-based incentives may also be beneficial to ensure a well-functioning code review process with high levels of collaboration.

```
Data: The initial set, IssuesSet, that includes all Issue events of the repository R that are
      opened within the initial assessment period = x days.
Result: Set of regularly opened Issues.
let ICD = [];
for (each issue in IssuesSet) do
    ICD.append(issues.creationDate());
end
D = [];
for (each adjacent pare ($d_1$, $d_2$) in ICD) do
    let timeDistance = ($d_2$ - $d_1$).days();
    D.append(timeDistance);
end
let IDM = D.median();
let waitTime = IDM;
while True do
    while the elapsed time since the last Issue of IssuesSet $<$ waitTime do
        if a new Issue is opened then
            IssueSet.append(new Issue);
            let d = time distance between the last two Issues of IssuesSet;
            D.append(d);
            IDM = D.median();
            waitTime = IDM;
        end
    end
    Send NewIssueNotification(R);
    waitTime = waitTime + IDM;
end
```

Algorithm 3: New Issue Notifier (NIN) where *ICD* is the set of Issues Creation Date, *D* is the set of temporal distances between Issues, and *IDM* is the Issue Distances median.

CHAPTER 6: CONVERSATIONS AND INTERCONNECTIONS IN TECH-RELATED ONLINE COMMUNITIES

Part of the content in this chapter was previously published in the Proceedings of the 2023 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2023) under the title: "How Popularity Shapes User Interaction in Tech-Related Online Communities."

Introduction

In recent years, online social networks have seen a significant increase in popularity and influence, allowing for forming and communicating within communities in new and unparalleled ways. The tech-based, socially-oriented online platforms have become widely used for communication, collaboration, and knowledge-sharing. The connectedness generated by these networks can be beneficial as it brings together people across many boundaries. One notable characteristic of these networks is the high degree of global collaboration among developers, particularly in developing open-source software.

The emergence of the internet has led to a shift in how users interact with information. They are no longer just consumers but also active contributors. They engage with one another, share their thoughts and ideas, and form online communities. The degree of interaction among users is one way to determine their social roles, which can be represented as positions, actions, or digital personas. These roles can be shaped within social networks formed in discussions within online communities, and they continue to evolve and change over time.

Determining roles within social networks has become increasingly important in recent times. For example, identifying the experts in a technical forum can make finding the most relevant answer to a question easier. Additionally, identifying individuals with a role in shaping the opinions of others is crucial, as it is based on their interactions and the spread of information through connections within the network. These individuals, who significantly impact the decisions made by a community, play a key role in accepting or rejecting preferences and trends. Therefore, examining these roles provides a deeper understanding and improved analysis of interactions within social communities. In online communities, users' status can significantly affect others' actions and attitudes. Those who are popular are often seen as influential figures, and their views and actions can shape the perspectives and behaviors of other members. When a popular user expresses a strong viewpoint or shares specific content, it can trigger a chain reaction of similar opinions and content among other users. This can impact the discussion topics, how they are approached, and the overall tone of the conversations.

The motivation behind this work was to study users' interactions, the characteristics of their social roles, and their commenting behaviors on online social communities. In this study, we aimed to collect data from GitHub repositories where individuals participate in projects that may require advanced algorithms, techniques, and strategies. These projects may lead to more in-depth and technical discussions among team members to achieve optimal solutions and evaluate outcomes. In order to obtain precise results, we also aimed to gather comparable data from Reddit and Stack Overflow platforms to establish a benchmark for cross-platform comparison of user behavior.

From this standpoint, our study seeks to answer the following research questions:

- **RQ1)** To what extent do users interact with tech-related social network posts?
- **RQ2)** Do users' popularity impact the overall commenting behavior of other users within the same community?
- **RQ3)** How are users involved in commenting communities based on their social features?

Method:

Dataset

For this research project, data was collected from three online platforms with integrated social features for communication, collaboration, and knowledge sharing among users. The platforms selected for data collection were GitHub, Reddit, and Stack Overflow, among the most widely utilized community-based online spaces, attracting millions of users. These platforms serve as virtual spaces that offer a vast and readily accessible repository of information for research purposes.

- **GitHub**: As mentioned in the first chapter, the Introduction, GitHub is a highly popular open-source platform utilized as a web-based hosting service for version control and collaboration between software developers.

  Each project hosted on GitHub has access control and collaboration features, including bug tracking, code review, and task management. Collaborative communication among GitHub users is facilitated through various channels, such as GitHub discussion forums and GitHub Issues.

- **Reddit**: is a popular conversation-driven social networking site that allows for asynchronous discussions among users on various topics [76]. Reddit had more than 50 million daily active users by late 2022 and 100 thousand active communities [77]. Reddit comprises many subcommunities called *subreddits* in which participants are presented with posts on a particular subject, interest, or theme.

- **Stack Overflow**: is a community-based Q & A website where the knowledge and expertise of programmers are leveraged to discuss and address coding issues through sharing different ideas and coding methods. In these informal environments, users can even provide code snippets suggesting a proper solution to technical coding-based issues. Stack Overflow has more than 100 million monthly users, and since its launch in 2008, over 50 billion developers have gotten help in resolving their technical queries [78].

We aimed to collect data related to the same tech-related areas in this work to obtain accuracy and perform a cross-platform comparison. We chose to retrieve data objects with the same interest in AI, machine learning, and robotics.

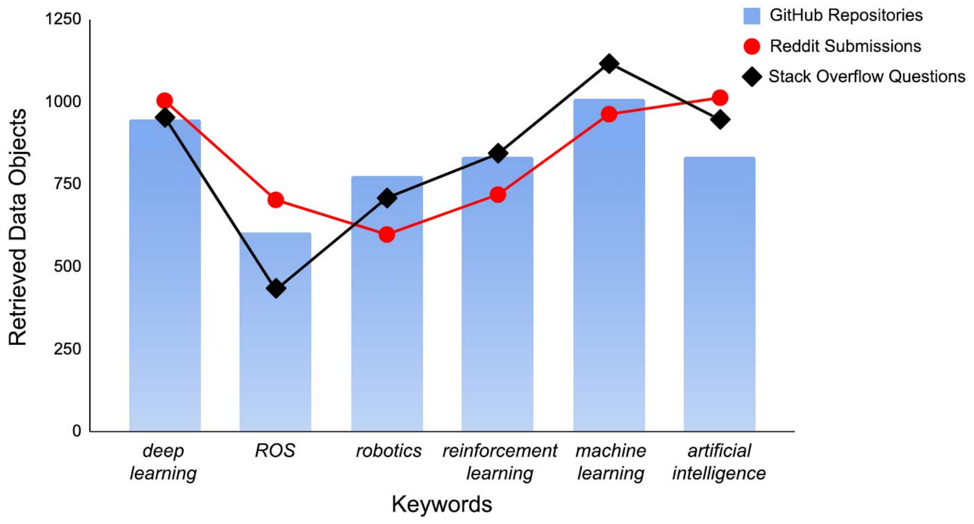


Figure 6.1: The number of retrieved *Posts* from GitHub, Reddit, and Stack Overflow grouped by the search keywords: "deep learning," "ROS," "robotics," "reinforcement learning," "machine learning," and "artificial intelligence."

For this purpose, we have developed a crawler using the Python programming language and utilizing the available REST APIs [70] [79] [80]. Using this crawler, we aimed to retrieve and filter data objects from GitHub repositories (including Issues), Reddit submissions, and Stack Overflow questions. We have utilized a set of keywords shown in Figure 6.1. In this work, for the reader's ease, we refer to GitHub Issues, Reddit submissions, and Stack Overflow questions as "*Posts*."

Given the numerous precautions in place to support the load balancing and throttling demands of the sites above, we had limitations on data request frequency. These constraints obstructed high-speed data gathering, and therefore, we had to limit the number of data objects we had to fetch to only 5,000 per platform. The dataset is available for download on the *Mega platform* at `https://bit.ly/abdul-dissertation-dataset` under the folder **Chapter06CommentingBehavior**.

The communication behavior of users was studied in this work, where users could interact with each other through the retrieved posts. The commenting mechanisms on online social networks let users leave comments in response to the introductory post or comment on other users' comments. The retrieved posts-related datasets contain different types of information, including data objects related to:

- Posts' **Authors**: (i.e., those users who initiate the conversations by creating Issues on GitHub, adding a submission on Reddit, or posting a question on Stack Overflow)
- **Commenters**: users who have submitted comments.

Table 6.1: Comment-related statistics extracted from the captured datasets.

	GitHub	Reddit	Stack OF
Avg. Comments Count per Post	17.24	6.38	7.29
Avg. Authors' Comments	5.53	1.46	1.50
Avg. Commenters Count	3.87	5.34	4.82
% One-Time Commenters	38%	89%	74%

Table 6.1 shows some comment-specific statistics extracted from the captured data objects where there is a small gap in the total number of comments on Reddit and Stack Overflow posts (6.38 and 7.29 comments per post). On the other hand, there was a relatively more significant average

number of collected comments on GitHub, where users left 17.24 comments on average per GitHub Issue.

Moreover, many of these comments across all three platforms are posted by the authors who have initiated the conversations. On average, 20-23% of the comments in Reddit and Stack Overflow are made by the posts' authors, while authors in GitHub showed a slightly higher rate of commenting, submitting 32% of all the collected comments.

GitHub Issues are primarily used to review the code, track bugs, and discuss new ideas. Hence, the more significant number of comments and the higher rate of authors' comments in GitHub compared to Reddit and Stack Overflow suggests a higher level of interaction between users.

Comment Timeline

The comment timeline is the sequence of comments that users submit over time for a post on an online platform. It may reveal various aspects of the time-related commenting behaviors. According to our research, GitHub commenters spent around four months on average between creating the Issue and its closure. This time interval is substantially larger on average when answering queries on Stack Overflow. On the other hand, the comment timeline period on most of the collected Reddit posts is considerably shorter than the other two platforms, where most comments are submitted over a few days.

Comment density: Comments are spread across the comment timeline, some very close to each other and some far apart. The comment density indicates how closely or widely dispersed the comments in a specific post's comment timeline are. The density of comments in open-source software platforms may be affected by project characteristics such as code size and project age [81], as well as other factors such as code functionality and code quality. Figure 6.2 depicts the comment timeline and the distribution of the daily captured comments for a randomly selected GitHub Issue.

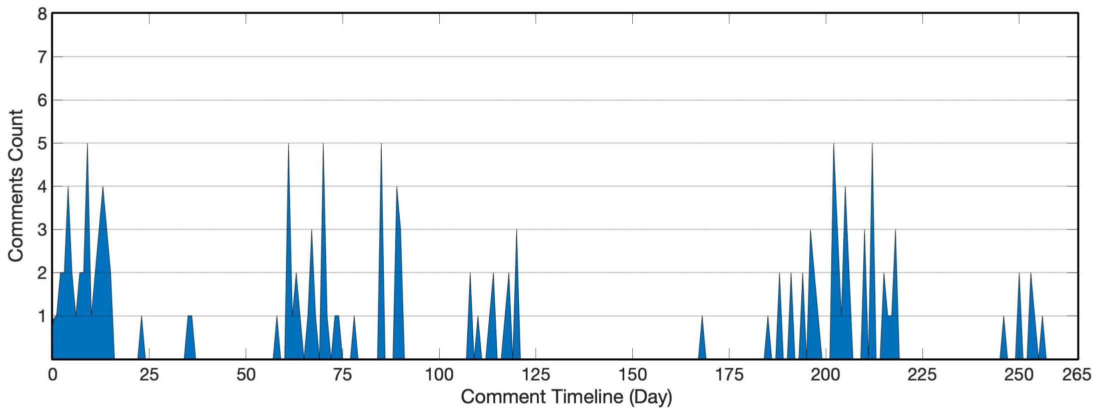


Figure 6.2: The figure shows the comment timeline of a randomly picked GitHub Issue, where it illustrates the comments counts per day throughout the Issue's lifetime.

Compared to clusters of comments with huge time gaps between them, comments posted in a short period may be more related. The more comments submitted in a short period, the more likely they will be related to direct and close conversations. When comparing the comment density across these platforms, it becomes understandable that people's overall engagement rate in leaving comments is higher due to the vast volume of daily Reddit posts. On GitHub, however, the data reveals a different behavior. Following creating an Issue on the GitHub repository, contributors begin discussing the intended activities, such as tracking new implementations, bugs, enhancements or code reviewing a pull request, by sharing their views and ideas.

Depending on the number of tasks and the code size, team members may need to spend more time discussing and communicating to accomplish their goals. Similarly, on Stack Overflow, other individuals (primarily developers) provide their answers when a user posts a question. Due to the nature of the commenting mechanism on Stack Overflow, other users may still post new answers supporting different points of view for a question that was posted a long time ago, providing a more efficient approach to solving the Issue.

Users Popularity

Users' popularity on social networks can be utilized as an indicator to measure how well they are observed, supported, and recognized by other users. To some extent, popular users on social networks have more credibility. Tech-related social networks have notoriously higher standards for sharing users' expertise and exchanging knowledge. Being more popular implies greater credibility, which supports users when they share their viewpoints in such communities. Building a reputation involves posting, sharing valuable content, and interacting with others. This work uses some extracted information to measure a user's popularity.

GitHub allows users to follow each other, much like most online networks. Through the following mechanism, followers can receive notifications embodying activities that have occurred by the users who have been followed. Users with more followers tend to have a more significant influence on other users, which can be seen as a characteristic of popular individuals [82]. In this work, we utilize the follower count to measure users' popularity on GitHub. In other studies, the number of followers was considered a proper measurement of user popularity [83].

Users' popularity can be measured on Reddit by *Karma*. Through the actions of "*upvoting*" or "*downvoting*," the users can vote on a submission or comment. Reddit uses an algorithm to calculate the *Karma* where the more upvotes a user receives, the higher the *Karma* will be. *Karma* encourages and enables community engagement, with users seeking to boost their score by interacting with others and sharing their opinions, knowledge, and expertise [84]. In this study, we use *authors' comment Karma* score provided via Reddit's API for each user to identify users' popularity.

Similarly, users earn rewards on Stack Overflow and gain some *reputation* through their activities [85]. Posting good questions and meaningful replies is the most effective strategy to acquire a reputation. Users gain (or lose) reputation based on how many people vote on their postings, which can give an insight into how the trust system works in Stack Overflow communities. Various studies on the Stack Overflow platform (e.g., [86]) suggest that *reputation* score is highly correlated to users' popularity.

Community Detection

This section aims to provide an approach to identify the communities of users in tech-related social networks. Social communities reveal various aspects of users' behaviors based on their characteristics and the connections derived from their interaction.

Social Roles Categorization

The level of interaction among individuals in social communities defines social patterns, which can be characterized as positions, behaviors, or virtual identities. In this section, "social role" is defined based on the users' social characteristics, which are related to the pattern of their behavior in online social communities and may not necessarily be directly related to a position or job title. These roles may be developed in social networks through direct communication and participation in discussions with others through forums, and they keep changing and evolving.

To explore users' social dynamics, we define four social roles in social networks with multiple communities based on users' popularity and users account age (since the account creation date):

- **Social Role 1 (SR1)**: is a role where most users have low popularity and low account age. This role may represent the new incomers.
- **Social Role 2 (SR2)**: indicates a social role in which users have accounts with a low age but

a high popularity. This role may indicate the relatively new users who attract more attention quickly and, hence, become popular in a short period.

- **Social Role 3 (SR3)**: displays the social role of users who have had accounts for a long time but have a low user popularity. One explanation is that even though users with such a social role do not attract others, considering that they have had their accounts for longer than others, this may indicate that these users are less active.
- **Social Role 4 (SR4)**: is a social role of the popular users that have been part of the specific social platform for a while. This role can indicate the experienced active users that have gained the confidence of others through their interactions.

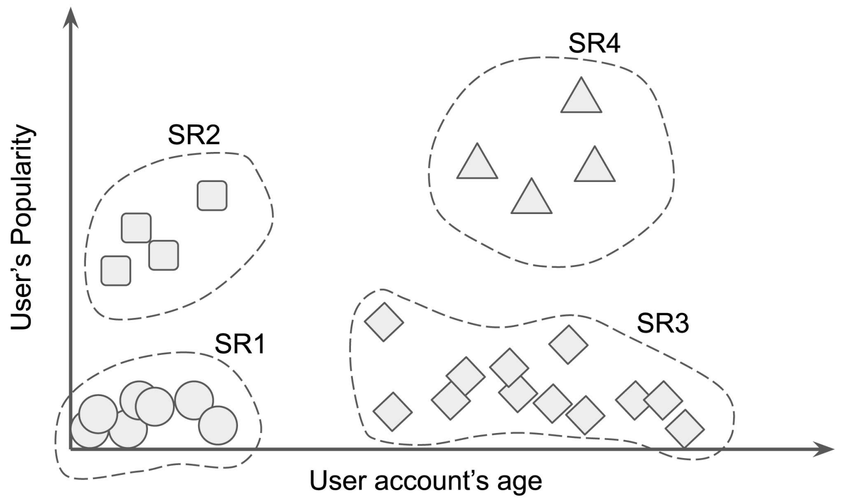


Figure 6.3: A sample plot of the commenters of a post, classified into four different social role classes based on the users' popularity and their account age, using the k-means Classifier.

We used the K-means Clustering [87] to categorize the commenters on each post. Figure 6.3 shows a sample GitHub Issue commenters classification using the K-means clustering based on users' popularity and account age. After the classification, the results are shown in Table 6.2 that most commenters across GitHub and Stack Overflow are of category SR3. At the same time, on Reddit, we observe a similar average number of commenters with the roles (SR1) and (SR3). On the other hand, across all three platforms, the commenters in a social role (SR2) are the minority.

In contrast, a much smaller number of users gained popularity relatively quickly. These results show the involvement of very popular commenters on almost every post, although users with the role (SR4) range between 14 and 22 percent across all platforms. This number can indicate that such communications benefit users with high seniority and popularity. Figure 6.4 depicts the distribution of commenters' count ratios based on their social role classification across all three platforms.

Table 6.2: The average ratio of the social roles that is categorized based on users' popularity and user account age.

	GitHub	Reddit	Stack OF
SR1	19.4%	38.2%	9.8%
SR2	5.0%	7.8%	2.2%
SR3	61.8%	36.5%	65.4%
SR4	13.8%	17.5%	22.6%

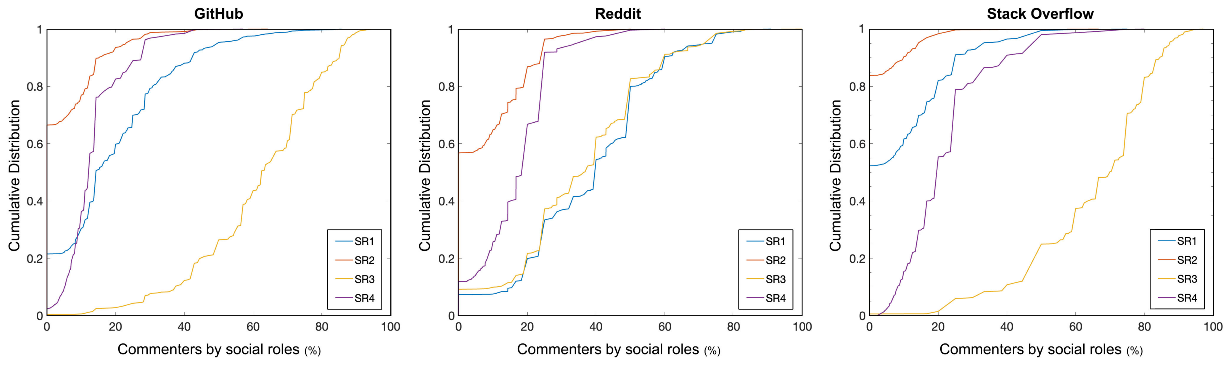


Figure 6.4: The cumulative commenters distribution based on their social role categories.

Commenting Behaviors

RQ1) To what extent do users interact with tech-related social network posts?

Figure 6.5 shows the distribution of the posts based on the comment counts. According to the illustrated data, on GitHub, over 82% of the captured Issues had 20 or fewer comments, whereas, on Reddit, nearly 95% of the submissions had eight or fewer comments. Similarly, 96% of the questions on Stack Overflow received only five or fewer comments. Moreover, we observed that there are a considerable number of posts that did not receive any comments (11% of the collected Issues in GitHub, 34% of the submissions on Reddit, and 14% of the questions in Stack Overflow) or that they only received one comment (17% in GitHub, 14% in Reddit, and 17% in Stack Overflow).

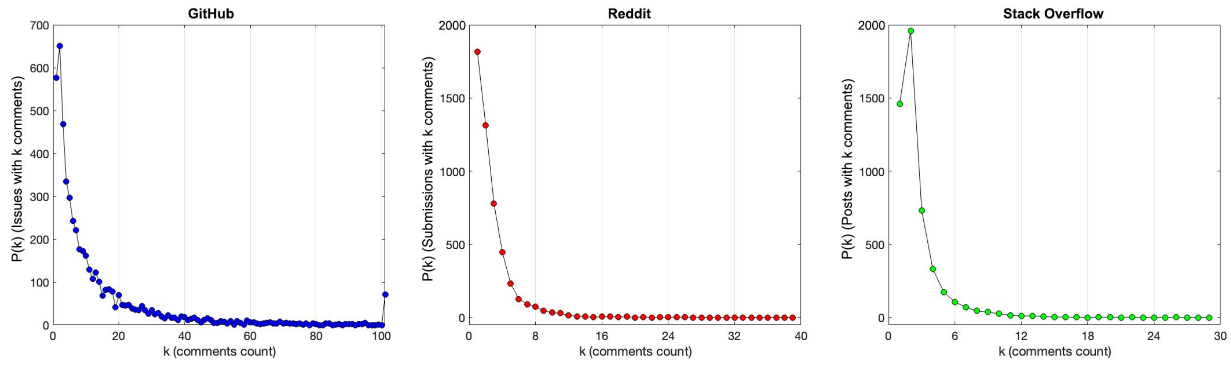


Figure 6.5: The distribution of the retrieved posts across all three platforms based on the total number of the extracted comments.

This trend may be associated with various communication factors that influence the level of participation in terms of attracting more or fewer contributors. Although these numbers may differ slightly in a more extensive data set, the pattern still shows that most posts receive only a few comments, and only a small number of posts receive a large number of comments. In other words, the users' engagements are not similar toward different posts of the same interest. This observation suggests that these posts follow the *Pareto principle* [88], which is one of the characteristics of scale-free networks (i.e., online social networks) [89].

Commenters can be divided into two groups based on the extent of their participation in commenting, regardless of any other characteristics:

- **One-time commenters**: Users who contribute by leaving only one comment on a post.
- **Multi-time commenters**: Users who comment more than once seem more active during a post's timeline.

The investigation shows that individuals comment on GitHub Issues an average of 4.45 times (the ratio of the average number of commenters over the comment count), indicating a significant number of multi-time commenters. Furthermore, approximately 38% of GitHub Issues users are one-time commenters. In contrast, this ratio is substantially higher on the other two platforms, where 89% and 74% of Reddit and Stack Overflow users, respectively, leave only one comment.

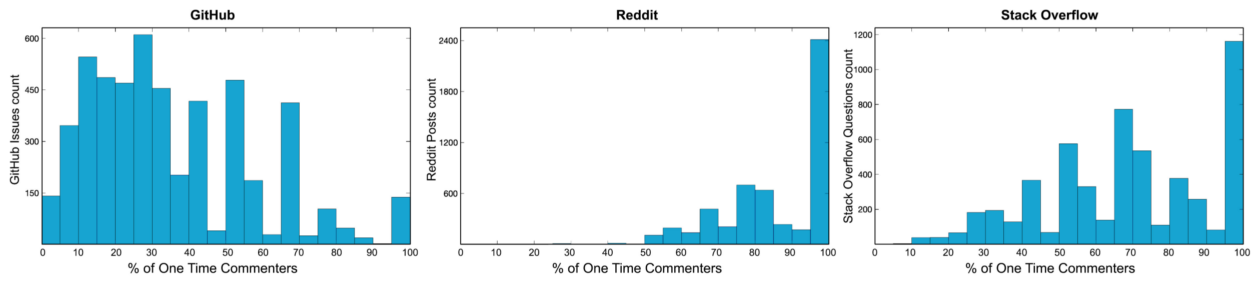


Figure 6.6: Illustrates distributions of the captured posts across all three platforms based on the ratio of the one-time commenters.

Figure 6.6 depicts the distribution of captured posts based on the comment-related ratio of their contributors. On GitHub, the distribution is right-skewed, as most of the Issues have a smaller ratio of one-time commenters, whereas many users comment multiple times. On the other hand, the distribution of Reddit posts reveals that users mostly comment once on most submissions. Even though Stack Overflow has a similar pattern where many of the captured questions obtain a higher rate of one-time commenters, we can observe that users tend to have multiple comments on more posts than on Reddit.

As a result, users' engagement through commenting on tech-related posts does not show a similar pattern on all social platforms. On social networks that involve groups of developers and reviewers working as a team (e.g., GitHub), the results suggest that users experience conversational behavior when communicating through posts. Such users utilize posts as discussion forums where conversations happen in online social settings to review others' works, share their viewpoints, and exchange information, as on GitHub. This observation is not seen on other social platforms like Reddit and Stack Overflow posts. For instance, compared to GitHub, most users engage with a post by leaving only a comment, indicating less conversational behavior.

Users Popularity Effect

RQ2) Do users' popularity impact the overall commenting behavior of other users within the same community?

To study the users' popularity and the potential impact that it can have on other communication-related features, we have investigated the popularity of two user categories: Authors: those users who initiate and submit the posts, and the most popular commenters or (*MPC*s).

Authors' Popularity Effect

Table 6.3 details the correlation of the author's popularity to the measurements: number of comments left on the same post, the total number of commenters that engaged with the post, the popularity of the commenters, and the author's comments count. We observe a weak positive correlation regarding the comments count, the relation between the authors' popularity, and the number of comments the posts they initiate receive. Even though this correlation is slightly stronger on GitHub and Stack Overflow, the average results showed that the more popular the authors get, the higher the likelihood of the posts they initiate receiving more comments. Consequently, as we see in Table 6.3, there is a weak but positive correlation between the popularity of the authors and the number of commenters who leave comments on all three platforms. A similar result was also derived in our previous work on GitHub Issues with a different data set [90].

Table 6.3: Author's popularity correlations to other comment-related measurements.

	GitHub		Reddit		Stack OF	
	r-val	*p-val*	*r-val*	*p-val*	*r-val*	*p-val*
Comments Count	0.30	0.01	0.12	0.00	0.39	0.02
Commenters Count	0.21	0.00	0.10	0.00	0.31	0.03
Author's Comment Count	-0.05	0.29	-0.03	0.49	-0.11	0.09
Commenters' Popularity	0.38	0.01	0.08	0.57	0.40	0.01

Moreover, regarding the authors' comments count, the results showed no clear correlation between authors' popularity and the number of times they comment on the same posts for all platforms.

We have also investigated the relationship between the authors' and commenters' popularity on the same posts. GitHub's and Stack Overflow's data show relatively strong correlations between the popularity of authors and commenters. Figure 6.7 shows the correlation between the popularity of the authors and the commenters of the same post. The observed pattern on GitHub and Stack

Overflow is that the more popular the authors are, the more likely they are to receive comments from popular users. This tendency, however, is not observed clearly on Reddit.

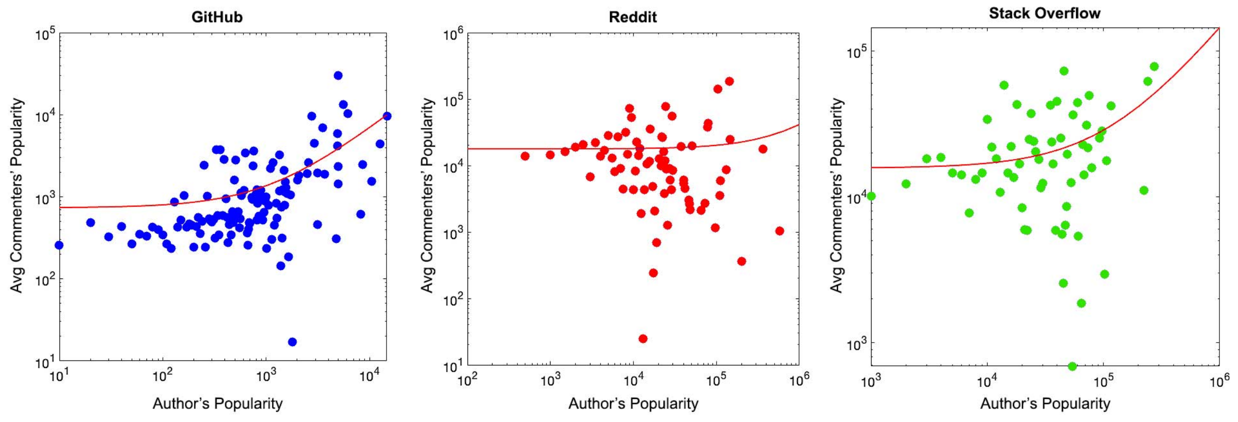


Figure 6.7: Depicts the log-log plot of authors' popularity correlation with the average commenters' popularity across the platforms.

MPCs' Popularity Effect

MPCs are the commenters with the highest popularity among other users who contribute to a post. Due to their popularity dominance and to study the potential impact on different commenting-related aspects or commenters' behavior, we investigated the most popular commenters, MPCs. The MPCs, as other commenters, may contribute to a post by commenting more than once. The captured data shows that 85% of GitHub posts include MPCs with multiple comments per post, which they contribute by leaving 27% of the total number of comments. On the other hand, on the other two platforms, MPCs tend to leave multiple comments relatively less. Most of the captured posts in Reddit and Stack Overflow (79% and 62%, respectively) have only one MPC contribution.

Contrarily to Reddit and Stack Overflow, it is implied that the most popular commenters' contribution on GitHub is considerably higher. This behavior suggests that MPCs are more involved in communications with other individuals who are engaged with the same posts.

MPC's Comment Tail: The comment tail is a concept we define to study any behavioral change in the number and the popularity of the commenters before and after the MPCs comment. Considering the comment timeline of the post (P) to include the comments $\{c_0, c_1, ..., c_k, c_{k+1}, ..., c_{n-1}, c_n\}$, ($n$) is the total number of the comments, (k) is the order of the MPC's comment, and $k \leq n$. Therefore, we define the comment tail to be equal to:

$$CT_P = \{c_{k+1}, c_{k+2}, ..., c_n\} \tag{6.1}$$

which is equal to a set of the post (P)'s comments starting from the comment at order (k+1) to the last comment at order (n).

As depicted in Figure 6.8 we define three types of comment tails based on their length:

- ***Short comment tails***: MPCs comment closer to the end of the comment timeline. In these conversations, a smaller number of individuals leave comments after the MPCs compared to the ones who comment before the MPC:

$$|CT_P| < |\{c_0, ..., c_{k-1}\}| \tag{6.2}$$

- ***Mid-range comment tails***: MPCs contribute closer to or at the middle of the conversation where a similar number of comments are observed before and after the MPC's comment:

$$|CT_P| \simeq |\{c_0, ..., c_{k-1}\}| \tag{6.3}$$

- ***Long comment tails***: MPCs tend to comment earlier in the conversation, which leads to a longer comment tail, and hence most commenters are engaged with the post after the MPC:

$$|CT_P| > |\{c_0, ..., c_{k-1}\}| \tag{6.4}$$

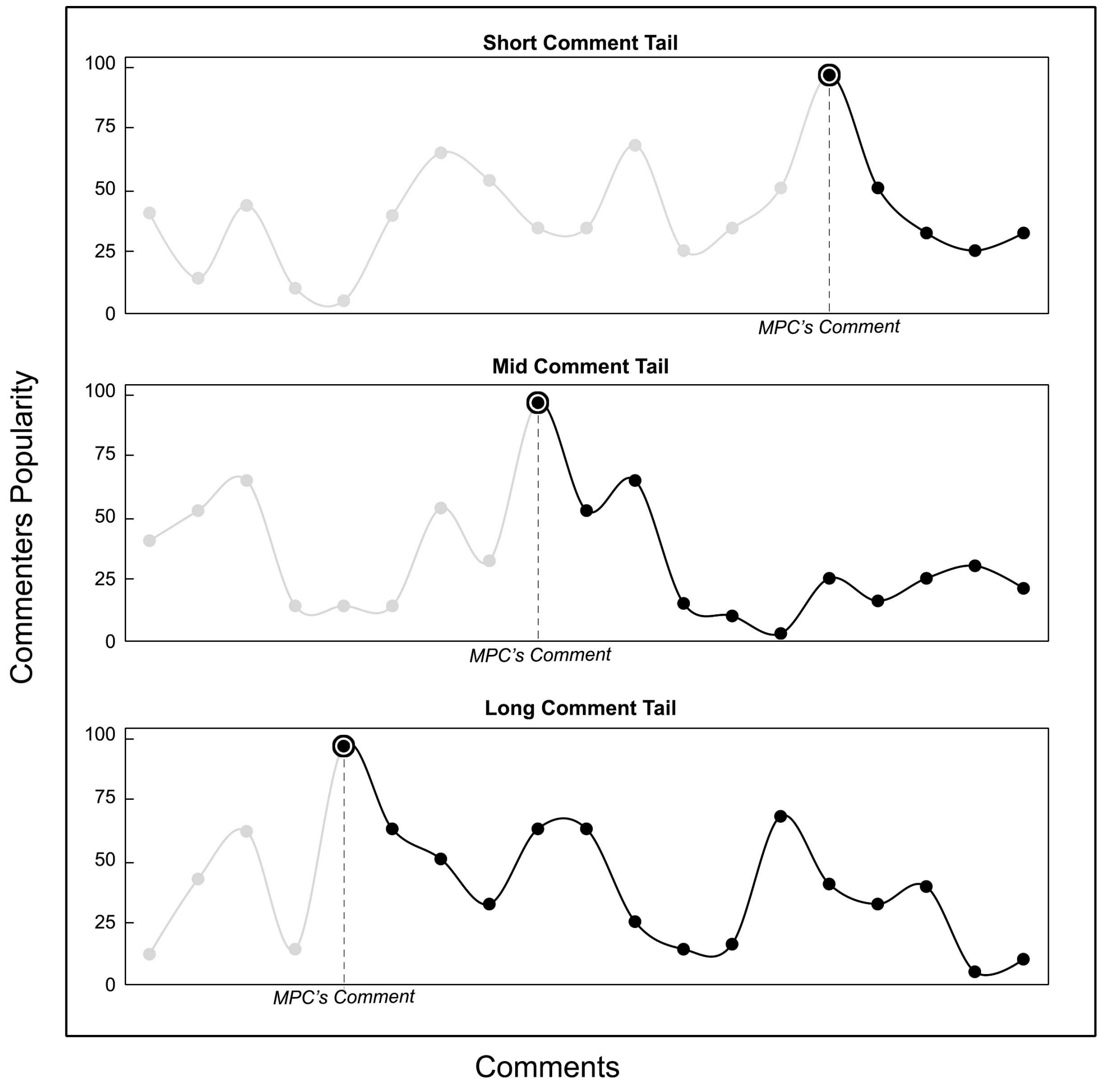


Figure 6.8: Illustrates 3 sample comment timelines, each showing one of the comment tail length categories: Long, mid, and short. The comment tail includes all comments that are posted after the MPC comments.

We have investigated the comments timelines and extracted each post's MPC-related comment tail to categorize them based on length. Table 6.4 details the ratio of each category and the commenting rate per user, including the authors, before and after the MPCs comment.

Table 6.4: MPC Comment tail statistics

	GitHub	Reddit	Stack OF
Short comment tail ratio	69.14%	40.39%	43.19%
Mid-range comment tail ratio	11.79%	20.56%	18.37%
Long comment tail ratio	19.07%	39.05%	38.44%
Comments per user before MPC	3.093	1.026	2.022
Comments per user after MPC	2.181	1.040	1.416
Authors' comments count before MPC	5.938	0.905	1.040
Authors' comments count after MPC	1.795	1.001	0.997

Based on the results, we observed a similar pattern of comment tail lengths on Reddit and Stack Overflow, with close numbers of comment tails of each length category. This observation shows no noticeable MPC-related change in users' comments count on Reddit and Stack Overflow.

In GitHub, however, more than two-thirds of the posts fall into short-length comment tails. In this case, we observe that the MPCs have commented mainly toward the end of the conversation. In other words, GitHub users' behavior of leaving multiple comments changes after MPCs' comments, where users show a lower tendency to comment. Similar to the users' commenting rate, there is a similar observation where the commenting rate of the authors becomes less after the MPCs' comments. In GitHub, we observe a more considerable change in authors' commenting rate, which may indicate a lower tendency of authors to comment after the MPCs leave their comments.

Figure 6.9 depicts the popularity distribution of different types of users. The data show that the posts on all platforms exhibit a similar pattern, where the popularity of the authors is slightly lower than most commenters. Additionally, from the commenters' popularity point of view, the users

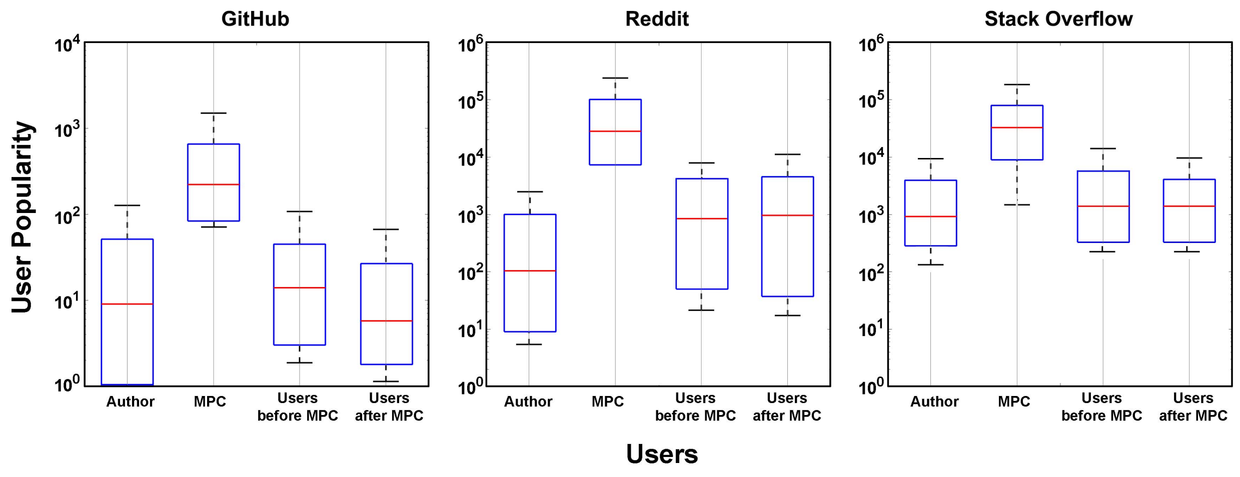


Figure 6.9: Shows users' popularity in three categories across the platforms: Authors, Commenters, and MPCs (the Most Popular Commenters).

who comment before or after MPCs mostly have similar levels of popularity on Reddit and Stack Overflow. However, on GitHub, the average popularity of the commenters decreases after the contribution of MPCs.

Table 6.5 details the correlation between MPCs' popularity and other users' popularity and the number of comments they post. Across GitHub and Stack Overflow, we observe a strong relationship between MPCs' popularity and the popularity of commenters regardless of their comment order. This correlation exists on Reddit, but it is slightly weaker. Furthermore, we could not observe a clear correlation between MPCs' popularity and authors' popularity across all platforms.

Table 6.5: MPC popularity correlations to other users' communications features.

	GitHub		Reddit		Stack OF	
	r-val	*p-val*	*r-val*	*p-val*	*r-val*	*p-val*
Comments Count	0.22	0.00	0.16	0.01	0.22	0.01
Authors Popularity	0.07	0.02	0.00	0.83	0.10	0.03
Commenters Popularity pre MPC	0.41	0.00	0.22	0.01	0.48	0.01
Commenters Popularity post MPC	0.35	0.00	0.11	0.00	0.33	0.04

From the total comments point of view, the correlation analysis shows a weak positive relationship between the popularity of the MPCs and the number of comments a post receives on all three platforms. The MPC popularity to comments count correlation indicates that it is likely for the posts to receive a higher number of comments if MPC's popularity is relatively high.

The observations from authors' popularity correlations reveal that even the number of comments received by different commenters on a post can be seen to be related to the nature of the posts. However, the results suggest that the contributions to a post in terms of commenting and the involvement of the users are correlated to the author's popularity, meaning posts that popular users initiate may involve a larger group of individuals and, hence, a more considerable number of comments in the discussion.

Furthermore, the MPC comment tail investigation results reveal changes in users' commenting rates and commenters' involvement before and after the MPC contribution in GitHub but not in Reddit and Stack Overflow. On the other hand, the results indicated that the MPC popularity positively correlates to other commenters' popularity, regardless of the platform. The posts that received comments from popular users will have a contribution from an MPC with relatively higher popularity than other posts. Similarly, the more popular MPCs are, the higher the likelihood of popular users' participation in the same post.

Social Commenting Communities

RQ3) How are users engaged in tech-related communities based on their social features? The social features of online coding platforms provide an environment for individuals to communicate with each other. When it comes to identifying the involvement of users, how their characteristics relate to their involvement, and understanding the users' overall communication behavior, we studied the users' commenting behavior through different posts based on their social roles. As a result, we built a weighted directed network where the nodes represent the users involved in commenting activities.

As it is depicted in Figure 6.10, the network has directed and weighted links where two nodes can be linked if at least one of the following conditions is met:

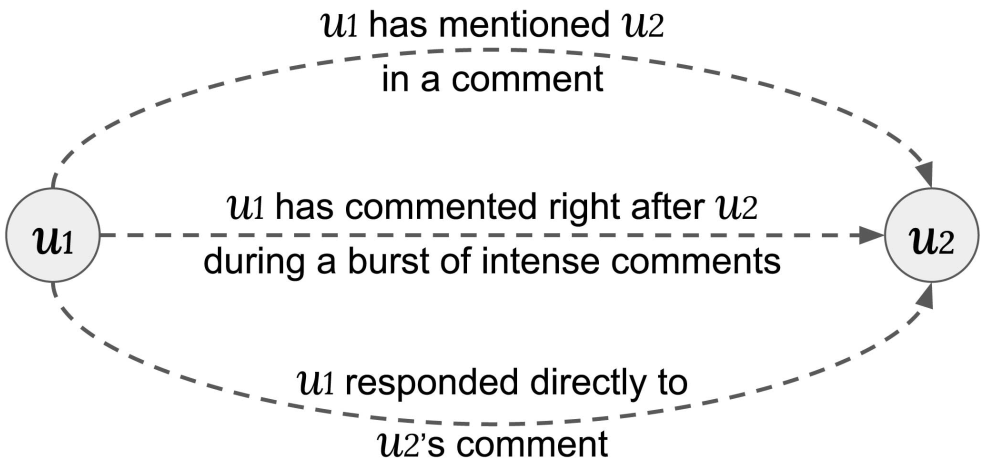

u_1 has mentioned u_2 in a comment
u_1
u_1 has commented right after u_2 during a burst of intense comments
u_2
u_1 responded directly to u_2's comment

Figure 6.10: Shows three conditions to define links in the commenting network. The nodes can have a direct weighted link through social tagging, conversation involvement, and directly responding to comments.

- Links by direct ***social tagging***: On social networks, the user social tagging refers to mentioning other users and linking them to a comment or any social post [91]. In this section, we utilize social tagging to identify users of a specific social platform inside the comments by searching for @mention tags or finding the mentioned valid usernames. Therefore, in this network, the node (u_1) will have a direct outgoing link to the node (u_2) if the latter is tagged in (u_1)'s comment.

- Links by being involved in ***conversation-like commenting***: The social interactions among peers through comments can be seen as a form of human-like dialog or conversation [40].

Throughout the commenting timeline, we observed bursts of intense comments, usually between several users who comment multiple times within a relatively short period. Such activities may indicate direct conversation behavior between team members through commenting, which defines this network's second type of link. Therefore, during such comments burst intervals, the node (u_1) will be linked directly to the node (u_2) if the former has commented right after (u_2) within a short period.

- Links by directly ***responding to comments***: Most social networks permit users to leave a direct comment, reply, or answer other users' comments. Hence, the node (u_1) is linked to the node (u_2) by an outgoing edge if the former has a direct comment on (u_2)'s comment.

Using our data set, we have constructed a commenting network for each platform based on the defined linking conditions. Each network shows the overall commenting activities of platform users across different posts, which includes subgroups or sub-communities of users who have direct interactions. In order to study the users' social interaction activities based on their roles, we have investigated these sub-communities within the more extensive commenting community. Hence, we have utilized the following three methods to detect the user commenting sub-communities:

- *Louvain*: It is a heuristic method to extract communities based on the density of the connections between the nodes [92]. This method has optimized the *modulairty* algorithm [93] and can detect communities, including those with fewer nodes.
- *Statistical Inference*: This method analyzes the properties of large networks to identify and uncover communities that have similar structural patterns [94].
- *Label Propagation*: This method, inspired by the spread of epidemics, detects communities by propagating node tags or labels based on the topological links between nodes until con-

vergence. Then, it uses the clustering structure of a graph to complete detecting communities where each community consists of nodes with the same label [95].

Table 6.6 shows the average number of nodes of the detected communities categorized by the detection method. According to the findings, compared to the other two approaches, the Louvain method returned comparatively larger communities regarding the number of users, where SR3 users predominate in each community.

Table 6.6: The detected community sizes (number of the nodes) by using the community detection method.

	GitHub	Reddit	Stack OF
Louvain	58	55	45
Statistical Inference	30	37	17
Label Propagation	32	10	12

Commenters Interconnection

The users within each detected community are connected through *weighted out-degrees* and *weighted in-degrees*. These weighted edges indicate the number of outgoing links (out-degree) and incoming links (in degree) from and to a node in a graph. In the commenting community, we see pairs of users who have exchanged more than one comment. Therefore, we have considered the links' weights to measure out-degrees and in-degrees [96]. For each platform's detected communities, in Figure 6.11, we visualized the ratio of the node-level measurements by users' social roles categorized by the community detection methods: Louvain [92], Statistical Inference [94], and Label Propagation [95].

Our investigation regarding the weighted out-degrees and in-degrees shows that all three commu-

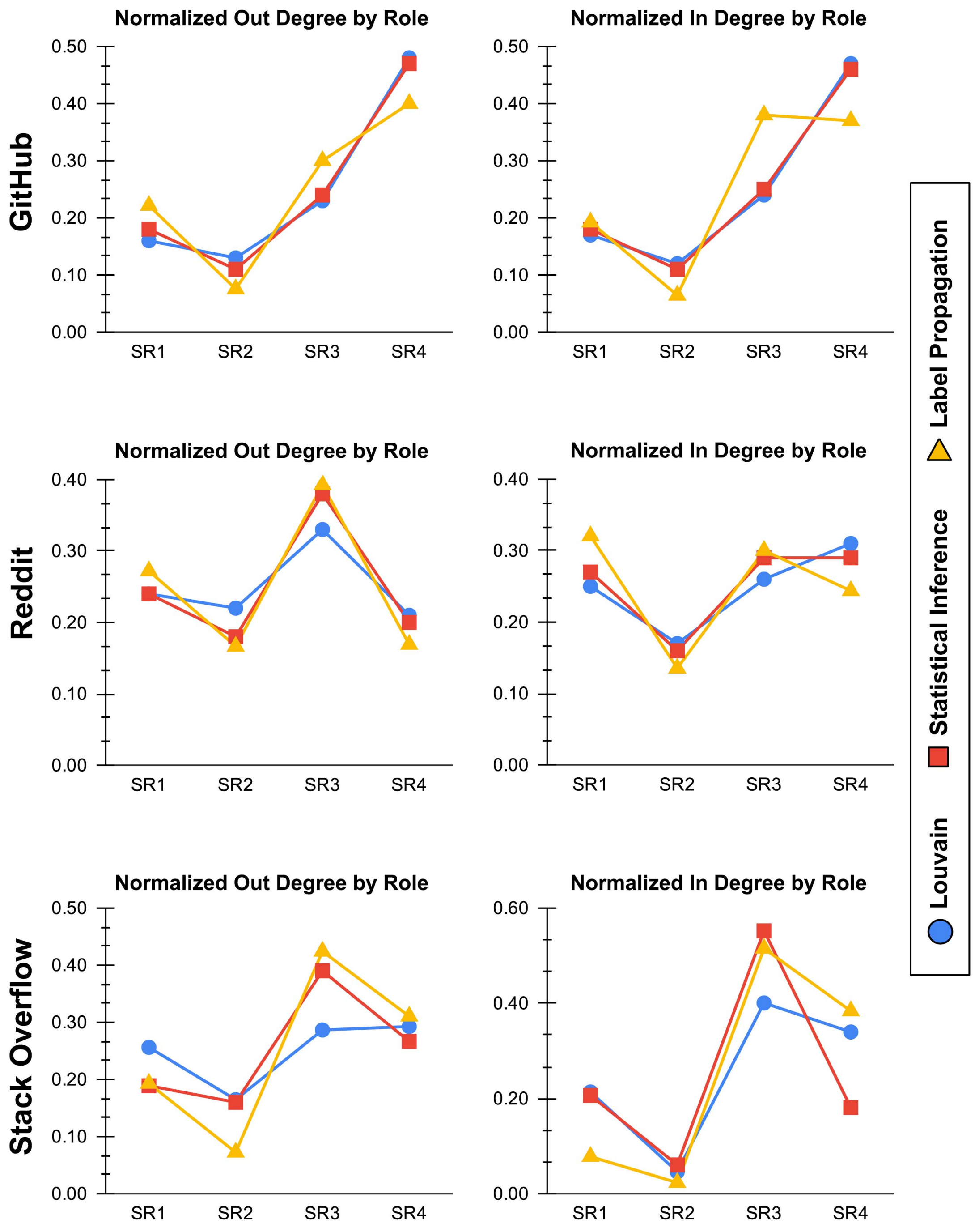


Figure 6.11: Depicts the constructed networks' node-specific measurements: weighted out-degree and weighted in-degree by social roles based on the user community detection methods: Louvain, Statistical Inference, and Label Propagation.

nity detection methods exhibit a relatively similar pattern in GitHub's commenting communities. In these communities, the SR4 commenters tend to leave more comments (out-degree links) while also being mentioned or tagged in others' comments or receiving direct comments from other users (in-degree links). Considering the total number of SR4 commenters (detailed in Table 6.2), their high level of weighted outgoing or incoming links highlights how active they communicate with other team members. However, regardless of the detection method used, commenters of social roles SR3 have a slightly higher rate of weighted out-degree and in-degree in most detected communities on Reddit and Stack Overflow.

Commenting Community Effective Engagement Score (CCEE)

From their social roles point of view, to understand how users are involved in the detected communities, we have investigated the following node-level measurements:

- *Betweenness Centrality*: The betweenness centrality measures how centrally a node is placed in a network based on connections to other nodes. Because of their control over how information is passed between nodes with a higher betweenness, nodes with a higher betweenness may significantly influence a network. Additionally, because they are located on the most significant number of message pathways, they are the ones whose removal from the network will have the most impact on communication between other vertices. The links' weights were considered while determining the betweenness centrality [96].
- *Clustering Coefficient:* The clustering coefficient measures the node's tendency to cluster with other nodes. In social networks, some nodes have a higher clustering coefficient because they connect to many other nodes. A node (u) with a higher clustering coefficient shortens the path between other nodes linked together indirectly via (u) [97].

Although a high value of betweenness centrality is the importance of that user in the community, it may not give a proper insight into how engaged the user is. Users with high betweenness centrality can sometimes be well-connected to others in a community. However, this is not always true where users are linked to a small number of users but yet highly centralized.

Users' engagement in communities may not be measured solely by relying on how centered a node is. Therefore, using the betweenness centrality and clustering coefficient measurements above, we have defined the Commenting Community Effective Engagement Score (CCEE) for nodes in commenting communities, which can be expressed as the following:

$$CCEE(u) = \beta(u)_c \delta(u)_c \tag{6.5}$$

where $\beta(u)_c$ is the betweenness centrality and $\delta(u)$ is the clustering coefficient of the node (u) in community c.

- The betweenness measurement $\beta(u)$ is computed using the following formula:

$$\beta(u)_c = \sum_{i,u,j \in c} \frac{\sigma_{ij}(u)}{\sigma_{ij}} \tag{6.6}$$

  where σ_{ij} represents the total number of the shortest paths connecting i to j, and $\sigma_{ij}(u)$ a subset of σ_{ij} and only includes the shortest paths that only pass through node u where i, u, and j all are nodes in commenting community c [98].

- The clustering coefficient $\delta(u)$ is calculated using the following formula:

$$\delta(u) = \frac{2t(u)}{k(u)[k(u)-1] - 2k^{\leftrightarrow}(u)} \tag{6.7}$$

  where $k(u)$ is the sum of in-degree and out-degree of node (u), $t(u)$ is the number of triangles through (u) that equals the total number of edges between the $k(u)$ neighbors of node (u), and $k^{\leftrightarrow}(u)$ is the bilateral degree of a node (u) and equals the sum of all the two-sided links between node (u) and its neighbor [99].

Since the detected communities are of different sizes, the $\beta(u)$ and $\delta(u)$ scale the community's size. Therefore, to compute these two measurements, we have normalized them to be in the range [0,1].

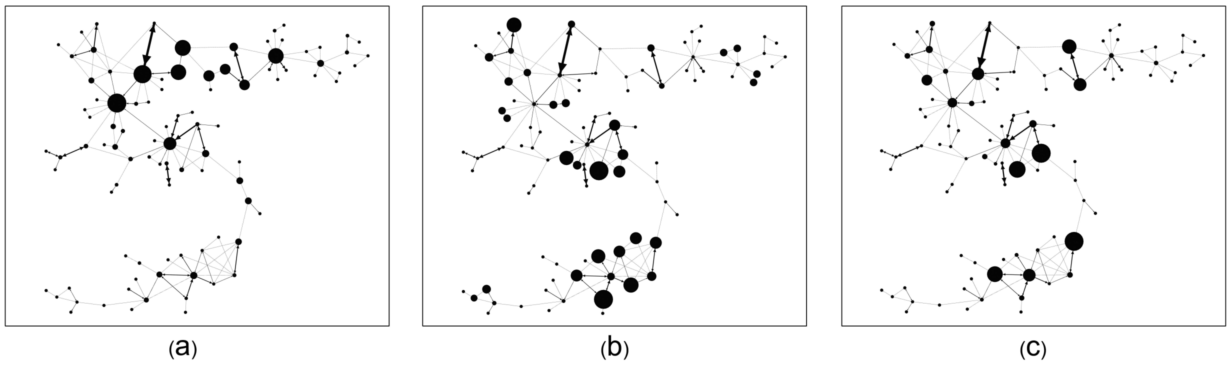


Figure 6.12: Illustrates a GitHub commenting community (detected using Louvain method) where nodes are sized based on three measurements: (a) the size of the nodes is proportional to the betweenness value, (b) the nodes' sizes indicating the value of the clustering coefficient, and (c) shows the nodes where they are sized based on CCEE score.

Figure 6.12 illustrates how nodes are sized based on nodes' betweenness centrality, clustering tendency, and CCEE score. This figure shows that users who are more significantly centralized and tend to be more clustered with others will get a higher effective engagement score. Figure 6.13 illustrates a detected GitHub commenting community using the Louvain method.

Figure 6.14 illustrates the average CCEE by social roles in all the communities detected using the abovementioned detection methods. Investigating users' engagements in GitHub and Stack Overflow platforms reveals a similar pattern of average CCEE score distribution based on users' social

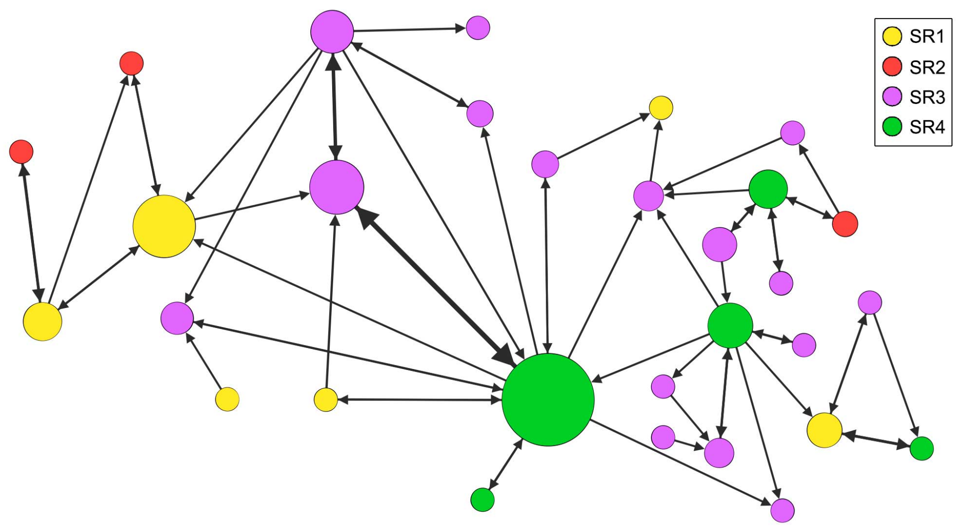


Figure 6.13: Illustrates one of the detected commenters communities in GitHub, which was uncovered using the label propagation method. The node size is relative to the CCEE score of the users in this sub-community.

roles. Regardless of the utilized community detection methods, the nodes representing SR4 commenters got the highest CCEE in these communities, indicating their involvement in commenting within their communities compared to other users.

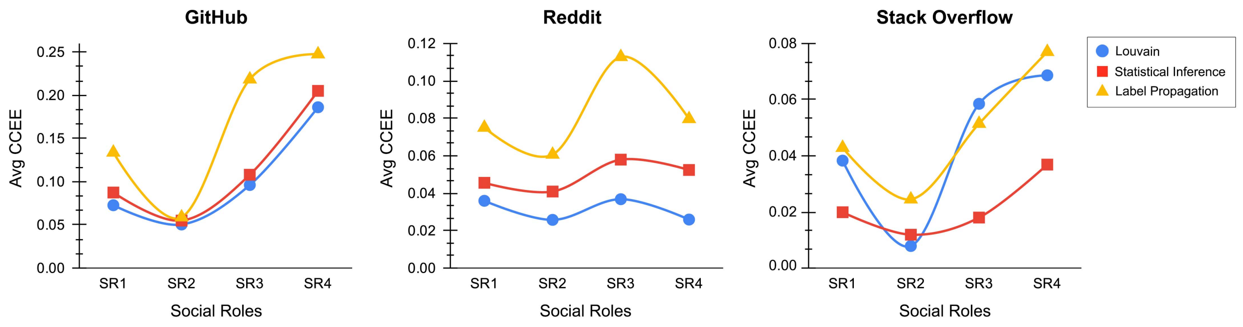


Figure 6.14: Shows the average commenting community effective engagement score (CCEE) categorized by social roles across all platforms' detected communities.

On the other hand, most of the commenting communities in Reddit, that were detected using

Louvain and *Statistical Inference* methods, do not reveal a significant difference in the degree to which users are engaged and involved in commenting. Very few communities show a slightly higher CCEE score for users with SR3 roles. However, when it comes to the communities that were detected by *Label Propagation* detection method, we observe numerous communities where the SR3 commenters have a higher calculated CCEE score.

As a result, we observe that the engagement positively correlates to users' social popularity and seniority represented by social roles, where popular users with older social accounts (indicating their seniority) are likely to be more engaged in communication with others through commenting. However, this correlation is more significant in GitHub and Stack Overflow, where the posts have more code-related content.

Conclusion

In this chapter, we aimed to analyze communication among users on online social networks. The focus was examining users' commenting behavior to understand interactions and conversations related to technical topics such as artificial intelligence, machine learning, and robotics. Data were gathered from three popular online platforms that offer social environments for user interactions: GitHub Issues, Reddit submissions, and Stack Overflow questions (all referred to as *posts* in this chapter).

This study aimed to assess the degree of engagement of individuals with technology-related posts on online social platforms. The results suggest that users on GitHub, where collaboration among developers and reviewers is a central feature, tend to engage more conversationally when communicating through posts. They utilize posts as discussion forums, where interactions take place in online social settings to review others' work, share perspectives, and exchange information.

However, this type of engagement is less prevalent on Reddit and Stack Overflow. A comparison of these platforms with GitHub reveals that the majority of users on these platforms engage with posts primarily by leaving comments, indicating a less conversational mode of engagement.

In this chapter, we also investigated the effect of users' popularity on the commenting behavior of other participants on the same post. To achieve this objective, the study evaluated the popularity impact of two user types: the authors of the posts and the most popular commenters (MPCs). The results of this study indicate that there is a noticeable degree of popularity-based impact on commenting behavior. Specifically, the data suggests that posts initiated by popular users tend to elicit more comments and involve a larger group of individuals in the discussion. In addition, by analyzing the impact of MPCs, the study reveals changes in users' commenting rates and involvement before and after the MPC's contribution on GitHub only. This pattern was not evident on Reddit and Stack Overflow. Furthermore, the results demonstrate that regardless of the platform, the participation of MPCs tends to attract more popular users to the conversation. These findings suggest that popularity plays a significant role in shaping the commenting behavior of users on online social platforms.

To gain a deeper understanding of user engagement in commenting on posts on online social networks, this study defined four social roles based on user characteristics. These social roles were determined by utilizing K-means clustering, considering users' popularity and account seniority. Additionally, a network was constructed to represent the interactions between users on each platform, and three community detection methods were employed to identify communities of commenters within the constructed networks. To evaluate the effectiveness of user engagement within communities of commenters, the study defined a metric known as the Commenting Community Effective Engagement (CCEE) score. This score aimed to identify users who are highly engaged in communication with their teammates and well-centered within their communities.

The results of this study revealed a positive correlation between engagement and users' social popularity and seniority, as represented by their social roles. Specifically, it was found that popular users with older social accounts tend to be more engaged in communication with others through commenting. This correlation was found to be more significant on GitHub and Stack Overflow, where the posts have a greater emphasis on code-related content.

Threats To Validity

The scope of our study was limited to three specific online social networks, and including data from other platforms could enhance the accuracy of our analysis. Additionally, our dataset was focused on a narrow range of topics within AI, robotics, and machine learning. A broader range of subject matter and an increased quantity of data could impact our analysis's validity. Furthermore, we only considered two user characteristics in determining social roles: popularity and account age, which may need to be revised to fully capture social roles across various online platforms.

CHAPTER 7: CONCLUSION

In this dissertation, we explored GitHub as a software development network. We delved into the dynamics of software development communities to provide an analytical work on the social nature of these communities, focusing on the diffusion of innovation, repository popularity, code quality, and users' commenting behavior. We aimed to offer valuable insights into how developers collaborated, shared knowledge, and innovated within one of the most popular open-source platforms.

Main Contributions

Modeling the Diffusion of Innovation in GitHub

In this dissertation, our initial focus was on uncovering the impact of GitHub's social system on adopting code. We accomplished this by investigating the influence of repository popularity. To do so, we employed an innovative methodology centered around popularity to represent the dissemination of innovation and predict code adoption, which we gauged by the frequency of fork events.

Our model utilized the relationships among GitHub users, repositories, and followers to construct timelines of events and social configurations. To simulate the spread of knowledge, we calculated how fork events were distributed across repositories and the network structure at three distinct time intervals after more fork events were integrated into the model from our dataset. Using this model, we assessed the degree to which past fork events (referred to as *popularity*) could predict future fork events (referred to as *innovation*).

We explored various social factors that influenced code adoption, including characteristics of repository owners, collaborators, and metrics such as star ratings, watchers, and forks. Our findings demonstrated the superiority of the popularity-based model in capturing the primary aspects of innovation diffusion: the *adoption of innovation* and *social interactions*. The popularity-based model we introduced in this section diverged from other models of innovation diffusion by placing a greater emphasis on the role of popularity rather than the network structure. While other models may have emphasized the significance of social networks and their interactions, our model treated network structure as a secondary outcome.

The popularity of GitHub repositories

Additionally, this dissertation delved into the dynamics of popularity and social engagement on the GitHub platform to gain a deeper understanding of how repository popularity evolved over time. We examined the impact of existing metrics such as forks, stars, and watchers on repository popularity.

To assess a repository's popularity accurately, we introduced the Weighted Total Popularity Score ($WTPS$), a weight-based score. To accurately measure the $WTPS$, we have utilized other popularity metrics of repositories considering the time factor. We modeled a community of GitHub repositories to derive the weightings, where a repository's popularity weight at a specific point in time was defined as the ratio of its gained popularity during that period to the total popularity gained by all other repositories during the same timeframe.

We also conducted a correlation analysis between $WTPS$ and other repository measurements. This analysis suggested that a higher $WTPS$ value was associated with a greater likelihood of the repository accumulating a more substantial number of stars than other popularity indicators, thereby enhancing its overall popularity score.

We also constructed a graph that represented the relationships between GitHub repositories and their owners' followers to study and evaluate choosing different popularity measurements for repositories. We conducted the popularity-based deletion process with $WTPS$, stars, forks, and watchers to compare their impact on the graphs. At each step, repositories with the highest popularity were removed from the graph to study their removal's impact on the remaining nodes' total clustering tendency. The results of node deletions showed similar results in the general graph's clustering coefficient when $WTPS$ and stars were the popularity indicators for the repositories.

However, because $WTPS$ is a weight-based score extracted from the history line of other popularity-related measurements, it can be considered a more accurate and reliable measure of repository popularity. Since social coding platforms' content and user population increase over time, it is better to have a consistent measurement that considers the time and is more robust to changes.

Code Quality in GitHub

In this dissertation, we also presented an analysis of the code review process when conducted using the GitHub Issue tracking mechanism. We examined how code review could be improved by analyzing the relationship between Issue frequency and community interaction, the involvement of experienced reviewers in Issues, and the correlation between the level of experience and the number of comments received by GitHub users.

We analyzed how various repository features influenced Issue frequency and timing. This analysis led us to propose a novel approach for improving the code review process, emphasizing regularity and community involvement. We introduced the New Issues Notifier (*NIN*), designed to encourage developers to maintain consistency in the review process by prompting them to open issues during dormant periods.

We established an initial period to collect sufficient Issue-opening patterns in this approach. Subsequently, we continuously calculated future time points based on the most recent set of Issues, notifying the team to initiate a new Issue. We simulated and executed this approach using three threads with different user acceptance probabilities. Our results demonstrated low user acceptance rates and increased regularity during the code review process.

To evaluate the effectiveness of this new approach, we designed a multi-layer network representing the Issue Community, linking users to the Issues they reviewed. From this network, we introduced a novel metric, the Issue Community Score (*ICS*), aimed at assessing community involvement and collaboration within the code review process facilitated by the issue tracking mechanism.

Although we believed that enforcing regularity was an essential aspect of the code review process, it did not necessarily yield greater community involvement since it was possible for repositories with high regularity to have a low *ICS* score. To gain a more robust Issue Community, contributors needed to have a higher interaction rate across many issues, which could lead to improved communication, knowledge spread, and thought exchange.

Encouragingly, even with the current GitHub Issue tracker, most repositories effectively recruited senior software engineers to participate in code reviews. Based on our research, offering additional recognition-based incentives could further enhance the code review process, fostering high levels of collaboration.

Commenting Behaviors in Social Communities of Developers

In this dissertation, we explored social network communication within tech-related communities on GitHub and compared it to similar communities on two other social platforms: Reddit and Stack Overflow. Our aim was to gain an understanding of how individuals interacted and to assess the level of engagement within online social communities, where users share an interest in technical topics such as artificial intelligence, machine learning, and robotics.

The results indicated that on GitHub, where developer and reviewer collaboration played a pivotal role, users tended to adopt a more conversational approach when participating in code review sessions. These sessions were effectively utilized as forums for discussion, fostering interactions in online social settings with the intention of reviewing peers' work, exchanging viewpoints, and sharing information. In contrast, this form of engagement was less commonly observed on Reddit and Stack Overflow. A comparison between these platforms and GitHub highlighted that the majority of users on Reddit and Stack Overflow primarily engaged with posts by simply leaving comments, signifying a less conversational style of interaction.

Additionally, we also explored how user popularity influenced commenting behavior on posts where we assessed the impact of two user types—post authors and highly popular commenters (MPCs). The findings revealed a noticeable popularity-based effect on commenting. Popular users' posts attracted more comments and a larger discussion group. Analyzing MPCs showed changes in user commenting activity before and after MPCs' contributions on GitHub, but this pattern was absent on Reddit and Stack Overflow. Moreover, MPCs attracted popular users to the conversation, underscoring the role of popularity in shaping users' commenting behavior on online social platforms.

This study established four social roles based on social characteristics, users' popularity, and account seniority, to better understand user engagement in online social network comments. In addition, we created a network to depict user interactions on each platform where we utilized three different community detection methods to identify communities of commenters. To assess user engagement in the commenting communities, we introduced the Commenting Community Effective Engagement score ($CCEE$). We showed users' popularity can impact their commenting behavior and involvement in tech-related social communities. The study findings demonstrated a positive correlation between user engagement and their social popularity and seniority, as indicated by their social roles. In particular, it was observed that individuals who are more popular within the community and have been active for a longer time tend to engage actively in communication through comments.

Potential Limitations

While this study provides valuable insights into the social dynamics of software development communities on GitHub, it is important to acknowledge its potential limitations.

One potential limitation is that the study focuses mostly on GitHub and does not explore other platforms or communities in depth. While GitHub is a popular platform, it may not represent all software development communities. Future research could explore other platforms to provide a more comprehensive understanding of software development communities.

Another potential limitation is that the study focuses on quantitative data and does not explore the qualitative experiences of developers, which may be subject to bias or inaccuracies. However, the study used a rigorous methodology to collect and analyze the data, which helped to mitigate this limitation. Additionally, the study proposes new metrics such as the $WTPS$ score, which may need further validation in future research.

Future Works

In light of the potential limitations outlined above, several recommendations emerge for guiding future research endeavors.

First and foremost, it will be valuable to broaden the scope of investigation beyond GitHub, encompassing various platforms and communities. This expanded perspective would enable a more holistic comprehension of software development communities, fostering a richer understanding of their dynamics and evolution.

Moreover, adopting a mixed-methods approach that combines quantitative data with qualitative insights can be considered for future research. Such an approach promises a more comprehensive exploration of the intricate social dynamics governing these communities.

Lastly, in potential future works, there might be a compelling need to investigate the experiences of underrepresented groups within software development communities, striving to uncover strategies for fostering diversity and inclusion. Simultaneously, a closer examination of the impact of cultural differences on these communities holds great potential, fostering cross-cultural collaboration and knowledge exchange within the global software development landscape.

Potential Applications

Despite the limitations, this study provides valuable insights into the social dynamics of software development communities on GitHub.

The research on social dynamics in software development communities on GitHub has several potential applications that can benefit developers, project managers, and tech organizations. Some real-world examples of how this study can be applied include:

- **Identifying Influential Developers:** By using the model to identify influential developers with a large number of followers or who contribute frequently to popular repositories, organizations can leverage their expertise to drive the adoption of new ideas and methodologies.

For example, a software development team could collaborate with influential developers to enhance the visibility and adoption of their projects on GitHub.

- **Improving Project Visibility:** Developers can use insights from the study to enhance the visibility of their projects on GitHub. For instance, by analyzing factors that drive repository popularity such as documentation quality, feature additions, and engagement with other developers, developers can strategically improve their project's visibility and attract more contributors and users.
- **Enhancing Code Review Process:** The study's findings on the relationship between issue frequency, community interaction, and reviewer involvement can be applied to improve the code review process on GitHub. For example, project managers can encourage regularity in the review process by nudging developers to open issues during specific periods based on previous behaviors, leading to more efficient and effective code reviews.
- **Promoting Collaboration and Knowledge Sharing:** Understanding social connections between developers can help organizations promote collaboration and knowledge sharing within their development teams. By analyzing user commenting behavior and engagement within online social groups, organizations can identify key influencers and encourage interactions that foster a culture of knowledge sharing and collaboration.

Additionally, the research on social dynamics in software development communities on GitHub can be leveraged to design data models, functions, and new bots, or even it can be utilized to enhance the current GitHub application in several ways:

- **Recommendation Systems:** By incorporating the findings on influential developers and repository popularity metrics, GitHub could develop recommendation systems that suggest popular repositories, influential developers to follow, or relevant projects based on a user's interests and activity. This can help users discover new projects, collaborate with influential developers, and stay informed about trending repositories.

- **Code Review Automation:** The insights from the study on the code review process and community interaction can be used to develop GitHub bots that automate certain aspects of the code review process. These bots could analyze issue frequency, reviewer involvement, and community interactions to provide feedback, suggest improvements, or streamline the review process for developers and project managers.
- **Social Collaboration Tools:** GitHub could integrate social collaboration tools based on the research findings to facilitate knowledge sharing, collaboration, and communication among developers. For example, features highlighting influential developers, promoting community engagement, or encouraging regularity in issue opening could enhance teamwork and foster a more interactive and supportive development environment.
- **Popularity Metrics Dashboard:** GitHub could implement a popularity metrics dashboard based on the proposed popularity score and repository characteristics identified in the study. This dashboard could provide developers with insights into their project's popularity trends, compare their repository's performance against others, and track the impact of their contributions on the platform.
- **Community Engagement Features:** Building on the study's insights into user commenting behavior and community involvement, GitHub could introduce features that encourage user engagement, such as badges for active community members, notifications for relevant discussions, or forums for sharing insights and best practices. These features can enhance community interaction and foster a sense of belonging among GitHub users.

LIST OF REFERENCES

[1] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72(2):027104, 2005.

[2] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. In *International Conference on Weblogs and Social Media*, volume 8, pages 361–362, 2009.

[3] David G Kleinbaum, Lawrence L Kupper, and Keith E Muller. Applied regression analyses and other multivariable methods. In *Applied regression analyses and other multivariable methods*, pages 718–718. 1988.

[4] George C Homans, A Paul Hare, and Richard Brian Polley. *The human group*. Routledge, 2017.

[5] Ferdinand Tönnies and Ferdinand Tönnies. *Gemeinschaft und gesellschaft*. Springer, 2012.

[6] Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

[7] Dan Ariely. Predictably irrational: the hidden forces that shape our decisions. *Math Comput Educ*, 44(1):68, 2010.

[8] Mark S Granovetter. The strength of weak ties. *American Journal of sociology*, 78(6):1360–1380, 1973.

[9] S Dixson. Number of global social network users 2017-2027. *Statistics*, 2023.

[10] Ichiro Kawachi and Lisa F Berkman. Social ties and mental health. *Journal of Urban health*, 78:458–467, 2001.

[11] Facebook. About facebook. `https://about.fb.com/`.

[12] Twitter. About twitter. `https://about.twitter.com/en_us/company.html`.

[13] Instagram. About instagram. `https://www.instagram.com/about/us/`.

[14] LinkedIn. About linkedin. `https://press.linkedin.com/about-linkedin`.

[15] Stack Overflow. About. `https://stackoverflow.com/company/about`.

[16] Reddit. About. `https://www.https://www.redditinc.com/about`.

[17] Quora. About. `https://www.quora.com/about`.

[18] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation of open source software developers. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 419–429. IEEE, 2003.

[19] Matthew Van Antwerp and Greg Madey. The importance of social network structure in the open source software developer community. In *2010 43rd Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2010.

[20] Shaosong Ou Alexander Hars. Working for free? motivations for participating in open-source projects. *International journal of electronic commerce*, 6(3):25–39, 2002.

[21] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*, pages 188–195. IEEE, 2013.

[22] Alexey N Medvedev, Renaud Lambiotte, and Jean-Charles Delvenne. The anatomy of reddit: An overview of academic research. *Dynamics On and Of Complex Networks III: Machine Learning and Statistical Physics Approaches 10*, pages 183–204, 2019.

[23] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 1277–1286, 2012.

[24] Abduljaleel Al-Rubaye and Ronaldo Menezes. Extracting social structures from conversations in twitter: A case study on health-related posts. In *Proceedings of the 27th ACM Conference on Hypertext and Social Media*, pages 5–13, 2016.

[25] Maurice Vergeer, Liesbeth Hermans, and Steven Sams. Online social networks and microblogging in political campaigning: The exploration of a new campaign tool and a new campaign style. *Party politics*, 19(3):477–501, 2013.

[26] Michael Bailey, Ruiqing Cao, Theresa Kuchler, and Johannes Stroebel. The economic effects of social networks: Evidence from the housing market. *Journal of Political Economy*, 126(6):2224–2276, 2018.

[27] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. Exploring the patterns of social behavior in GitHub. In *Proceedings of the International Workshop on Crowd-based Software Development Methods and Technologies*, pages 31–36, 2014.

[28] Zhewei Zhang, Youngjin Yoo, Sunil Wattal, Bin Zhang, and Rob Kulathinal. Generative diffusion of innovations and knowledge networks in open source projects, 2014.

[29] Ramya Akula, Niloofar Yousefi, and Ivan Garibay. Deepfork: Supervised prediction of information diffusion in github. *arXiv preprint arXiv:1910.07999*, 2019.

[30] Jirateep Tantisuwankul, Yusuf Sulistyo Nugroho, Raula Gaikovina Kula, Hideaki Hata, Arnon Rungsawang, Pattara Leelaprute, and Kenichi Matsumoto. A topological analysis of communication channels for knowledge sharing in contemporary github projects. *Journal of Systems and Software*, 158:110416, 2019.

[31] Daron Acemoglu, Asuman Ozdaglar, and Ercan Yildiz. Diffusion of innovations in social networks. In *2011 50th IEEE Conference on decision and Control and European Control Conference*, pages 2329–2334. IEEE, 2011.

[32] Samer Faraj, Sirkka L Jarvenpaa, and Ann Majchrzak. Knowledge collaboration in online communities. *Organization science*, 22(5):1224–1239, 2011.

[33] Hudson Borges, Marco Tulio Valente, Andre Hora, and Jailton Coelho. On the popularity of github applications: A preliminary note. *arXiv preprint arXiv:1507.00604*, 2015.

[34] Hudson Borges, Andre Hora, and Marco Tulio Valente. Predicting the popularity of github repositories. In *Proceedings of The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, pages 1–10, 2016.

[35] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the factors that impact the popularity of github repositories. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–11, 2016.

[36] Joicymara Xavier, Autran Macedo, and Marcelo de Almeida Maia. Understanding the popularity of reporters and assignees in the Github. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, pages 484–489. Knowledge Systems Institute Graduate School, 2014.

[37] GitHub.com. Trending: See what the github community is most excited about today. `https://github.com/trending`, Accessed on 2023-03-05.

[38] GitHub Awards. Discover your ranking on github. `https://github.com/vdaubry/github-awards`. [Online].

[39] Git most wanted. `http://gitmostwanted.com/`. [Online].

[40] Stefano Pace, Stefano Buzzanca, and Luciano Fratocchi. The structure of conversations on social networks: Between dialogic and dialectic threads. *International Journal of Information Management*, 36(6):1144–1151, 2016.

[41] Samaneh Saadat, Chathika Gunaratne, Nisha Baral, Gita Sukthankar, and Ivan Garibay. Initializing agent-based models with clustering archetypes. In *Proceedings of the International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*, Washington, D.C., July 2017.

[42] Giuseppe Destefanis, Marco Ortu, David Bowes, Michele Marchesi, and Roberto Tonelli. On measuring effects of github issues' commenters. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, pages 14–19, 2018.

[43] Haoxiang Zhang, Shaowei Wang, Tse-Hsun Chen, and Ahmed E Hassan. Reading answers on stack overflow: Not enough! *IEEE Transactions on Software Engineering*, 47(11):2520–2533, 2019.

[44] Subhasree Sengupta and Caroline Haythornthwaite. Learning with comments: An analysis of comments and community on stack overflow. In *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[45] Daejin Choi, Jinyoung Han, Taejoong Chung, Yong-Yeol Ahn, Byung-Gon Chun, and Ted Taekyoung Kwon. Characterizing conversation patterns in reddit: From the perspectives of content properties and user participation behaviors. In *Proceedings of the 2015 ACM conference on online social networks*, pages 233–243, 2015.

[46] Mathilde Forestier, Anna Stavrianou, Julien Velcin, and Djamel A Zighed. Roles in social networks: Methodologies and research issues. *Web Intelligence and Agent Systems: An International Journal*, 10(1):117–133, 2012.

[47] Eric Gleave, Howard T Welser, Thomas M Lento, and Marc A Smith. A conceptual and operational definition of social role in online community. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–11. IEEE, 2009.

[48] Saya Onoue, Hideaki Hata, and Ken-ichi Matsumoto. A study of the characteristics of developers' activities in GitHub. In *Asia-Pacific Software Engineering Conference*, volume 2, pages 7–12, 2013.

[49] Cody Buntain and Jennifer Golbeck. Identifying social roles in reddit using network structure. In *Proceedings of the 23rd international conference on world wide web*, pages 615–620, 2014.

[50] Howard T Welser, Eric Gleave, Danyel Fisher, and Marc Smith. Visualizing the signatures of social roles in online discussion groups. *Journal of social structure*, 8(2):1–32, 2007.

[51] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165, 2014.

[52] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment? *Information and Software Technology*, 74:204–218, 2016.

[53] Yao Lu, Xinjun Mao, Zude Li, Yang Zhang, Tao Wang, and Gang Yin. Does the role matter? an investigation of the code quality of casual contributors in github. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 49–56. IEEE, 2016.

[54] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt,

vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201, 2014.

[55] Aron Fiechter, Roberto Minelli, Csaba Nagy, and Michele Lanza. Visualizing github issues. In *2021 Working Conference on Software Visualization (VISSOFT)*, pages 155–159. IEEE, 2021.

[56] Zhifang Liao, Dayu He, Zhijie Chen, Xiaoping Fan, Yan Zhang, and Shengzong Liu. Exploring the characteristics of issue-related behaviors in github using visualization techniques. *IEEE Access*, 6:24003–24015, 2018.

[57] Dorota Celińska and Eryk Kopczyński. Programming languages in github: a visualization in hyperbolic plane. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 11, pages 727–728, 2017.

[58] Shishir Dubey, B Balaji, Dinesh Rao, Deepak Rao, et al. Data visualization on github repository parameters using elastic search and kibana. In *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 554–558. IEEE, 2018.

[59] Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. A study of external community contribution to open-source projects on github. In *Proceedings of the 11th working conference on mining software repositories*, pages 332–335, 2014.

[60] Everett M Rogers. *Diffusion of Innovations*. Simon and Schuster, 2010.

[61] GitHub.com. The State of the Octoverse 2017.

[62] Georgios Gousios. The ghtorent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 233–236. IEEE, 2013.

[63] GitHub.com. Event types & payloads.

[64] Kevin Peterson. The GitHub open source development process, 2013. http://kevinp.me/github-process-research/github-processresearch.

[65] Albert-Lazlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[66] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440, 1998.

[67] Albert-László Barabási and Eric Bonabeau. Scale-free networks. *Scientific American*, 288(5):60–69, 2003.

[68] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.

[69] Diomidis Spinellis. *Code quality: the open source perspective*. Adobe Press, 2006.

[70] GitHub. Github rest api.

[71] GitHub, 2020.

[72] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[73] GitHub. Classifying your repository with topics.

[74] Steven Loria. textblob documentation. *Release 0.15*, 2:269, 2018.

[75] SmarterBear. The state of code review 2020 report, 2020.

[76] Katie Elson Anderson. Ask me anything: what is reddit? *Library Hi Tech News*, 2015.

[77] Reddit, 2021.

[78] Stack Overflow.

[79] Reddit. Reddit api.

[80] StackExchange. Stack exchange api v2.3.

[81] Oliver Arafat and Dirk Riehle. The comment density of open source software code. In *2009 31st International Conference on Software Engineering-Companion Volume*, pages 195–198. IEEE, 2009.

[82] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. Understanding the popular users: Following, affiliation influence and leadership on github. *Information and Software Technology*, 70:30–39, 2016.

[83] Abduljaleel Al-Rubaye and Gita Sukthankar. Scoring popularity in github. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 217–223. IEEE, 2020.

[84] Annika Richterich. ' karma, precious karma!'karmawhoring on reddit and the front page's econometrisation. *Journal of Peer Production*, 4(1):1–12, 2014.

[85] Dana Movshovitz-Attias, Yair Movshovitz-Attias, Peter Steenkiste, and Christos Faloutsos. Analysis of the reputation system and user contributions on a question answering website: Stackoverflow. In *2013 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM 2013)*, pages 886–893. IEEE, 2013.

[86] Arpit Merchant, Daksh Shah, Gurpreet Singh Bhatia, Anurag Ghosh, and Ponnurangam Kumaraguru. Signals matter: understanding popularity and impact of users on stack overflow. In *The World Wide Web Conference*, pages 3086–3092, 2019.

[87] Pádraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers - a tutorial. *ACM Computing Surveys*, 54(6):1–25, jul 2022.

[88] Albert-László Barabási. The new science of networks. *Cambridge: Perseus*, 2002.

[89] Lubos Takac and Michal Zabovsky. Data analysis in public social networks. In *International scientific conference and international workshop present day trends of innovations*, volume 1. Present Day Trends of Innovations Lamza Poland, 2012.

[90] Abduljaleel Al-Rubaye and Gita Sukthankar. Improving code review with github issue tracking. *arXiv e-prints*, pages arXiv–2210, 2022.

[91] Michele Zappavigna and James R Martin. # communing affiliation: Social tagging as a resource for aligning around values in social media. *Discourse, context & media*, 22:4–12, 2018.

[92] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[93] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of sciences*, 103(23):8577–8582, 2006.

[94] Lizhi Zhang and Tiago P Peixoto. Statistical inference of assortative community structures. *Physical Review Research*, 2(4):043271, 2020.

[95] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[96] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social networks*, 32(3):245–251, 2010.

[97] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[98] Ulrik Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social networks*, 30(2):136–145, 2008.

[99] Giorgio Fagiolo. Clustering in complex directed networks. *Physical Review E*, 76(2):026107, 2007.