What is the Gist? Understanding the Use of Public Gists on GitHub

by

Weiliang Wang
B.Sc., Southeast University, 2013

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

What is the Gist? Understanding the Use of Public Gists on GitHub

by

Weiliang Wang
B.Sc., Southeast University, 2013

Supervisory Committee

Dr. Daniel M. German, Supervisor
(Department of Computer Science)

Dr. M. Member One, Departmental Member
(Department of Same As Candidate)

Dr. Member Two, Departmental Member
(Department of Same As Candidate)

Dr. Outside Member, Outside Member
(Department of Not Same As Candidate)

Supervisory Committee

Dr. Daniel M. German, Supervisor
(Department of Computer Science)

Dr. M. Member One, Departmental Member
(Department of Same As Candidate)

Dr. Member Two, Departmental Member
(Department of Same As Candidate)

Dr. Outside Member, Outside Member
(Department of Not Same As Candidate)

ABSTRACT

Abstract Here.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

DEDICATION

Just hoping this is useful!

Chapter 1

Introduction

GitHub[1] is a very popular source code hosting site which serves as not only a collaborative coding platform but also a social media. Various unique features of GitHub has greatly facilitate developers' collaboration, communication and coordination during a project, which has given rise to many research in different angles.

However, *Gist*[2], as a very important feature of GitHub, has been paid little attention by researchers, and there's no existing gists dataset available, which makes the GitHub gists a brand new research area. Therefore, a lot of questions on GitHub gists remain unanswered, and my research would focus on gists in this thesis. Gist is a unique feature on GitHub allowing users to easily share snippets and pastes with others. Every GitHub user can create any number of gists. GitHub explains Gist as follows:

"Gists are snippets of code. They can be entire applications, or they can just involve a single file. Best of all, every gist is a git repository, which means that they can be ***forked****,* ***cloned****, and manipulated in every way."*

To my knowledge, Gist has gain much interests since more and more GitHub users start using it. However, what the whole picture of gists looks like still remains unclear. For example, how many GitHub users are using gists? Do users fork/commit/comment other users' gist? Gists were originally designed for people's sharing code snippet, but we don't know what kind of code snippets those users are sharing. For example, how many files are there in a gist? What kind of languages are gists mainly using? How many lines of code are there in a gist? How many methods are defined in a gist? All these questions need to be answered. Also, apart from gists

[1] `https://www.github.com`

[2] `https://help.github.com/articles/creating-gists`

of code snippets, there may exists other types of files in gists, and what these files are about is also unknown.

What's more, gists could be utilized in many non-trivial ways which could be out of people's realization. Some applications have been developed to help users better manage and share gists, such as *GistBox*[3], *Gisto*[4]. *WordPress*[5] supports embedding a gist either by its URL or its ID.[6] There are also some examples showing that GitHub gists can be used in many other ways, not just for containing some codes. Carl Sednaoui wrote a blog[7] about how to make best to-do list by using a private gist. GitHub user Jeremi M Gosney used gist to write a blog[8] about how he obtained the private key for `www.cloudflarechallenge.com`. All these threads show that gists are not only used for code snippets hosting and sharing.

Therefore, I propose the following research questions in this thesis:

- *What do gists look like?* Although gists are becoming more popular among GitHub users, we still don't have a whole picture of gists. For example, how many of all GitHub users are using gists? What are the most popular languages used in gists? What is the size of a gist and how many files are contained? Do users collaborate on gists? All these sub-questions come down to one research question: what on earth do gists look like?

- *How do people use gists?* GitHub gists were initially developed for users to write and share code snippets, but gists could be used in many other ways. Thus it's worth to find out different ways of using gists which could provide good recommendations and suggestions on GitHub gists could be better used.

In the following part of my thesis, the writing would be in XX folds:

[3] `http://www.gistboxapp.com/`
[4] `http://www.gistoapp.com/`
[5] `http://www.wordpress.com/`
[6] `http://en.support.wordpress.com/gist/`
[7] `http://carlsednaoui.com/post/70299468325/the-best-to-do-list-a-private-gist`
[8] `https://gist.github.com/epixoip/10570627/`

Chapter 2

Related Works

2.1 Background of GitHub

For the large and complex software projects, it is a challenge to well coordinate the collaboration between all the developers and to manage each developer's contribution. Based on this demand, Version Control System (VCS) was developed to record changes to a file or set of files over time so that you can recall specific versions later. [1] The traditional Centralized Version Control System (CVCS) requires developers to commit a change to a central repository, merge and resolve the conflicts. However, it has some severe downsides in that the entire history of the project lies in the central database. If the central database becomes corrupted, everything would be lost. [1][2]

Distributed Version Control System (DVCS) relaxes the requirement of CVCSs to have a central, master repository. Each developer owns a first-class repository of its own right, containing the entire project history.[2] DVCSs make developing, releasing and coordinating large open source software projects much more flexible than traditional CVCS. [3]

Among all these DVCSs, Git has gained the most momentum. It began its life in 2005 as a revision management system used for coordinating the Linux kernal's development. Over the years, Git has evolved by leaps and bounds due to its functionality, portability, effciency, and rich third-party adoption.[4]

Based on Git, some Web-based applications have been developed to host open source projects for free and to make it easy for project management using a Web-based user interface. It's convenient for developers to set up repositories, clone existing projects and commit their contributions.[4] Moreover, such applications lay much

emphasis on the social aspect of software engineering.

Among all these social medias, GitHub is the most known one. GitHub users all have an account. They can not only broadcast their activities to others who are interested but also follow other users or projects they are interested in. Besides, users can explore some projects they might be interested in, share snippets and pastes with others, and write some blogs. By bringing these social networking features to DVCSs, it's of great help for communication and coordination.[5]

Generally speaking, GitHub is a social coding website, combining the social networking features and software engineering. Such a new environment gives rise to lots of research in different angles. Due to the social aspects introduced to GitHub, problems become more complicated. Sometimes both quantitative and qualitiative methods should be used for analysis. It could not be a good way to organize this literature review according to methodology, since the same problem could be analyzed using different methods. Therefore, the framework of this literature review is built on different research purposes.

2.2 Recent Research on GitHub

GitHub created a new open source environment that has given rise to lots of research in different angles. GitHub is such a data pool that contains not only developers' profiles but also numerous projects, as well as developers' activities such as their contribution to projects and their interactions with other developers.

To make full use of such a great amount data, more and more researchers are trying to discover some findings or patterns in terms of software engineering. Some of them looked into developers' profile and their activities, attempting to help employers find technical experts on GitHub.[6][7][8][9] Also, some researchers focused on the source code in project repositories. For example, Bissyande et al. (2013) took advantage of rich data on GitHub by conducting a large scale and comprehensive study that examine the "popularity", "interoperability", and "impact" of various programming language measured in different ways, such as lines of code, development teams, issues, etc.[10] Apart from that, some researchers also tried to discover patterns in developers' collaboration and interation, such as how developers assess each other and find proper partners[11][12][13], the herding phenomena on GitHub[14], the relation between developers' behavior on GitHub and other Q&A websites like

StackOverflow[1][15], etc.

However, no researchers have ever conducted studies around GitHub gists, which makes it a brand new research spot.

2.3 GitHub Gists

2.3.1 Related Tools

Taking the advantage of strong user base, GitHub gists has been widely used, based on which a lot of third-party tools have been developed. For example, a Chrome App, *GistBox*[2], was developed to help users better organize GitHub gists and save snippets seen on the web. *sublime-github plugin*[3] allows users to save and share a snippet of code to GitHub gists without leaving Sublime. *jist* is a command-line utility for managing multi-file, multi-directory private gists. [4] *gist-it* helps users quickly and easily share the code to GitHub gists.[5].

Actually there already exist many snippets management tools similar to GitHub gists like *pastebin*[6], *snipt*[7], *codepen*[8], *dabblet*[9], etc. However, GitHub gists have some unique features that make it more convenient to use and collaborate. Over these years, GitHub gists keep being updated to make it more user friendly as is stated in their blogs[10].

2.3.2 The Features of Gists

As for the features of GitHub gists, Wikipedia provides a good explanation stated as follows[11].

"Gist builds upon that idea by adding version control for code snippets, easy forking, and SSL encryption for private pastes. Because each 'gist' is its own Git repository,

[1] http://www.stackoverflow.com/
[2] http://www.gistboxapp.com/
[3] https://github.com/bgreenlee/sublime-github
[4] http://charlesleifer.com/blog/jist-a-command-line-utility-for-managing-multi-file-multi-dire
[5] https://atom.io/packages/gist-it
[6] http://pastebin.com/
[7] https://snipt.net/
[8] http://codepen.io/
[9] http://dabblet.com/
[10] https://github.com/blog/1837-change-the-visibility-of-your-gists; https://github.com/blog/1850-gist-design-update
[11] http://en.wikipedia.org/wiki/GitHub#Gist

multiple code snippets can be contained in a single paste and they and be pushed and pulled using Git. Further, forked code can be pushed back to the original author in the form of a patch, so pastes can become more like mini-projects."

All these features make GitHub gists very functional and user friendly, helping users better manage their gists and share with others.

Chapter 3

Methodology

3.1 Research Questions

In Chapter 1, we proposed two main research questions to be studied. This part will further demonstrate them and try to divide them into several sub-questions as well as their value and importance.

3.1.1 What do gists look like?

This is a very general question, which needs to be answered from multiple aspects. Thus we should divide this question into small questions as listed below.

- How many of all GitHub users are using gists?

  GitHub gists was developed several years after GitHub was launched. Although there have been a great number of GitHub users, we still don't know how many of them use gists. This question could help us to have an idea of the popularity of gists among all GitHub users.

- How many gists does one user usually own?

  This helps to understand if users use gists much or not.

- How many files are there in a gist?

  A GitHub gist is usually assumed to host short code snippets, but we still have no idea how many files there usually be in one gist.

- How big is one gist usually?

  GitHub gists are known to be usually in a small size since they are used to host code snippets, which could be verified by the answer to this question.

- Is there a relationship between files included in one gist?

  Many gists contain more than one file. There must be a reason why users would put multiple files in one gist, which reflect that there must exist some kind of relationship between files. By answering this question, we can reveal these kinds of relationship.

- Does there exist other types of files than source code files?

  Although gists was created for developers to write and share source code snippets, but it does not stop some users using gists in a non-trival way. This question may help us find more ways of using gists that other users even have never thought of.

- What are the most popular languages used in gists?

  We also want to know whether there is a pattern among all the popular programming languages used in gists.

3.1.2 How do people use gists?

Similarly, if we want to understand how people use gists, it might be as well to divide this question into several small questions as listed below.

- Do users discuss much on GitHub gists?

  Users can make comments under other people's gists, which makes it more easier for developers to share thoughts and discuss isssues. However, do they really make full use of this feature to make much discussion with other people? It remains to be answered.

- Do users collaborate on gists?

  GitHub gist has very unique features which are version control and easy forking for code snippets. Because each gist is its own Git repository, it can be pushed and pulled using Git. Further, forked code can be pushed back to the original author in the form of a patch, so pastes can become more like mini-projects.

This machenism greatly facilitates collboration between developers. But it remained unclear whether there is also an active application of such machenism on gists.

- What are people's purposes of using gists?

  It's known that gists are mainly used to write and share source code snippets. However, the purpose of doing so must be various which needs to be summarized.

Both qualitative and quantitative methodology were performed to understand GitHub gists by answering the questions demonstrated above.

3.2 Data Collection

This part describes the data collection from GitHub. All the acquired data would be used to understand the content of gists. Since the goal is to get a whole picture of the gist usage, any gists of any GitHub users would be of interest.

A rich collection of GitHub data can be accessed through REST API provided by GitHub, but there still exist some challenges, mainly because the data is too huge and its scheme is not documented. Therefore, in order to make it convenient enough for developers or researchers to acquire or analyze these GitHub data, Gousios et al. developed *GHTorrent*[1], which is a project aiming to offer a mirror of GitHub's data and event streams to the research community as a service in a scalable manner.[16] It provides dataset of GitHub users, repositories, commits, comments, etc in either MySQL or MongoDB format, saving researchers' a lot of time of downloading the data.

To the end, we leveraged GHTorrent to obtain the users dataset (the MongoDB version in 2014-03-29[2]) including all GitHub users (2,592,527 users totally). We did some filtering to the users data based on two aspects of consideration. First, there may exist duplicated users in the dataset. Second, there may exist unreal users in the dataset. We determine a user is unreal according to two characteristics: one is that it has a NULL value for the *ext_ref_id* attribute in the users dataset of MySQL version, and another is that the user login is composed of eight random uppercase letters. After filtering out the unreal users and duplicates, 2,407,094 users were left.

[1] `http://www.ghtorrent.org/`

[2] `http://ghtorrent.org/downloads.html`

Unfortunately, GHTorrent does not provide dataset of gists, which means we have to gists information by ourselves. GitHub API[3] was leveraged to query the data. GitHub API is based on the REST architecture, which uses pull strategy for data retrieval and users should explicitly make requests to collect data. Despite the adaptability of GitHub API, there are some constraints and limitations of using it. GitHub describes the rate limiting as follows: "For requests using Basic Authentication or OAuth, you can make up to 5,000 requests per hour. For unauthenticated requests, the rate limit allows you to make up to 60 requests per hour."[4] Thus, in order to get a higher query speed, each request for data retrieval must be signed with valid GitHub user tokens.

Instead of downloading the gists files, we tried to use GitHub API to first query the gists metadata. Based on our research questions, we focused on the information in the metadata in the following aspects:

- gist identifier
- identifier of the gist owner
- description, which describes what this gist is about
- languages used, not only programming languages could be included
- number of files contained
- size in Byte
- content
- times of being forked
- number of being committed and history of these commitments
- number of being commented
- created date
- updated date
- type, such as text/image/application etc.

[3] `https://developer.github.com/v3/`
[4] `https://developer.github.com/v3/#rate/limiting`

In order to guarantee the representativeness of the targeted users, all github users were first randomized using a Perl script, then we started the query. To be able to use GitHub API, we used a Python script to keep making a HTTP Request, parsing the obtained data to json format and storing them into json files in the mapping skema of "user: gists metadata". The query stopped when we successfully obtained gists metadata of 200,000 users. Table 3.1 shows the distribution of these users based on their gists.

Users	**Number**	**Percentage**
Having at least one gist	11,680	5.84%
Having no gists	188,320	94.16%

Table 3.1: Distribution of obtained users

Since there is only a small number of github users use gists, another week was spent querying gists metadata. At last, we obtained metadata of totally 618,393 gists, which would be used for quantitative analysis.

As for the real content of gists content, we went through all the gists mentioned above and tried to write the content of each gists file into one separate file being stored locally. At last we successfully wrote **793,891** files to the local machine, which would be used for quantitative analysis of gists files.

3.3 Quantitative Analysis of Gists Metadata

Based on the obtained gists metadata, we made some simple statistical analysis, trying to figure out the characteristics of gists and get a overview of what gists look like. We manipulated the metadata of the obtained 618,393 gists, trying to answer the following questions:

- How many gists does each user own?
- How many files are there in a gist?
- How large is a gist?
- What kinds of programming/non-programming languages are used in gists?
- How many commits/forks/comments are there in a gist?

- How many days are there between the created date and most recent updated date for each gist?

For each of these aspects, we tried to answer the questions through statistical distribution presented by figures and tables.

3.4 Analysis of Gists Content

After having a good understanding of gists metadata, we started looking into the real content of these gists. We did quantitative analysis and qualitative analysis respectively to gain a further comprehension of gists content.

3.4.1 Quantitative analysis of gists content

As mentioned above, we wrote each gists file into the local machine, and tried to make quantitative analysis to these files.

- Analyzing source code files

  First, we intended to analyze the content of gists files written in programming languages with the help of the source code analyzing tool *CLOC*[5]. This tool can count blank lines, comment lines, and physical lines of source code in many widely used programming languages[6].

  Taking the advantage of this tool, we calculated the number of code lines, comment lines and blank lines for all files in each gist, as well as the maximum value, minimum value, mean value, median value, mode value, standard value and variance value of number of lines.

  Apart from that, in order to see whether there is a difference in results between different programming languages, we also performed the same calculation for specific languages.

- Analyzing gists files not indicating a programming language

  Then we analyzed the gists files written in non-programming languages. We tried to better understand these gists by looking into the "type" field in gists

[5] `http://cloc.sourceforge.net/`
[6] `http://cloc.sourceforge.net/#Languages`

metadata. After that, for each "type" of gists, we extracted the extension of gists files from filename to acheive a further understanding.

3.4.2 Quanlitative analysis of gists content

In order to make quanlitative analysis of real contents of gists, we randomly chose 400 gists from all the obtained gists and tried to manually analyze them. For each gist, we did coding for its content in terms of "relationship" and "content".

- Content:

  From the aspect of content, we can divide gists into several categories. We tried to manually assign label(s) to each gist based on gist description, key words in gist files and personal experience. These labels are not necessarily exclusive, including: (1) Code: any kinds of source code; (2) Test: test code; (3)Class: only a class is defined; (4) Template: coding example/pattern; (5) Command: commands used in shell; (6) Function: only one or several functions are defined; (7) fragment: only one or several lines of code without complete functions/class definitions or implementations; (8) Note: script other than source code; (9) Log: log files, or error messages after running code; (10) Configuration: config files used for build/run code; (11) Diff: diff files used for visualizing the changes in a file; (12) Documentation: text tutorial documentations; (13) Data: data stored in json, csv or other forms; (14) Blog: technical blog usually written in markdown files; (15) Non-technical: notes without any technical content.

- Topology:

  Although most gists only contain one file, but there still exist many gists containing more than one gists. We assume there must exist some topological relationship between these files in one specific gist. These relationship could be: (1)Siblings: files in a gist are independent; (2) Reference: one file refers to another file or calls functions/methods defined in another file; (3) Generation: one file is the input/output of another file; (4) Test: one file is to test the code of another file; (5) Attachment: one file contains the configuration/readme information of another file.

After finishing the data coding, we made statistics to the number of occurance of each label, so that we could have a clear idea of what gists are about.

3.5 Quanlitative Analysis of How People Use Gists

As is known, Twitter has become a big information gathering pool, providing much data for researchers. Recently, Twitter has come to be a more and more popular platform for developers to learn, share, discuss technical questions, so we chose Twitter as a information source to help understand how people are using gists. We used "github gist" as the searching keywords and read all the searched tweets one by one. While reading these data, we tried to classify them into several categories of how people use gists.

Chapter 4

Findings

4.1 FINDING 1: About 5.8% of GitHub users use gists.

Table 3.1 shows that there are 5.84% of GitHub users use gists. In order to verify that this proportion applies to the whole picture, we queried two other samples, each including gists metadata of 200,000 users. Users distribution is shown in table 4.1 and 4.2.

Users	**Number**	**Percentage**
Having at least one gist	11,595	5.80%
Having no gists	188,405	94.20%

Table 4.1: Distribution of obtained users

Users	**Number**	**Percentage**
Having at least one gist	11,631	5.82%
Having no gists	188,369	94.18%

Table 4.2: Distribution of obtained users

The results shown in table 4.1 and table 4.2 are consistent with the one of table 3.1, so that we can conclude that there are around 5.8% of users using gists (having at least one gist).

4.2 FINDING 2: Most gists users only have a small number of gists.

Figure?? and table 4.3 show the proportion of our targeted users owning different number of gists. We can see that majority of users just have a small number of gists. In our targeted users, 70.66 % have less than 10 gists, and each targeted user has at most 30 gists.

Number of Gists One User Owns	Percentage of Users
1	21.38%
2	12.24%
3	8.71%
4	6.64%
5	5.34%
6	4.44%
7	3.69%
8	3.16%
9	2.75%
10	2.31%
11	2.14%
12	1.86%
13	1.63%
14	1.40%
15	1.38%
16	1.22%
17	1.08%
18	1.06%
19	0.99%
20	0.92%
21	0.87%
22	0.82%
23	0.84%
24	0.82%
25	0.95%
26	1.05%
27	1.22%
28	1.67%
29	2.63%
30	4.81%

Table 4.3: Percentage of GitHub users who have different number of gists

4.3 FINDING 3: Majority of gists only contain one file.

We can see obviously from figure **??** and table 4.4 that majority of gists (86.79%) contain only one file. Actually we thought there should be just several files in a gist, since gists are just used to record and share small code snipts. However, one important observation we obtained is that, unexpectedly, there also exists gists containing many files (1001 files in a gist at most) which are really of interest for further study.

Number of Files Each Gist contains	Percentage of Gists
0	0.02%
1	86.79%
2	7.16%
3	3.16%
4	1.53%
≥ 5	1.35%

Table 4.4: Distribution of gists containing different number of files

4.4 FINDING 4: Gists are usually small.

No matter from the "size" field of metadata, or from the number of lines calculated by CLOC tool, or from the manual analysis of the gist content, we can always infer that majority of gists files are very small. These three aspects are demonstrated one by one as follows.

4.4.1 Size

Figure 4.1 and table 4.5 show that the distribution of "log2(size)-number of gists" is very similar to normal distribution, and the size of majority gists assemble in a narrow range (around 1 kB), which means majority of gists have a small size. However there also exists very large gists, which is out of expectation. Thus we think it is necessary to dig into details about these gists with large size.

log_2(**size of a gist in bytes**)	**Percentage of Gists**
0	0.02%
1	0.04%
2	0.15%
3	0.22%
4	0.57%
5	1.63%
6	4.36%
7	9.62%
8	16.42%
9	20.55%
10	18.73%
11	13.27%
12	6.91%
13	3.29%
14	1.60%
15	0.95%
16	0.64%
≥ 17	1.04%

Table 4.5: Distribution of gists having different size

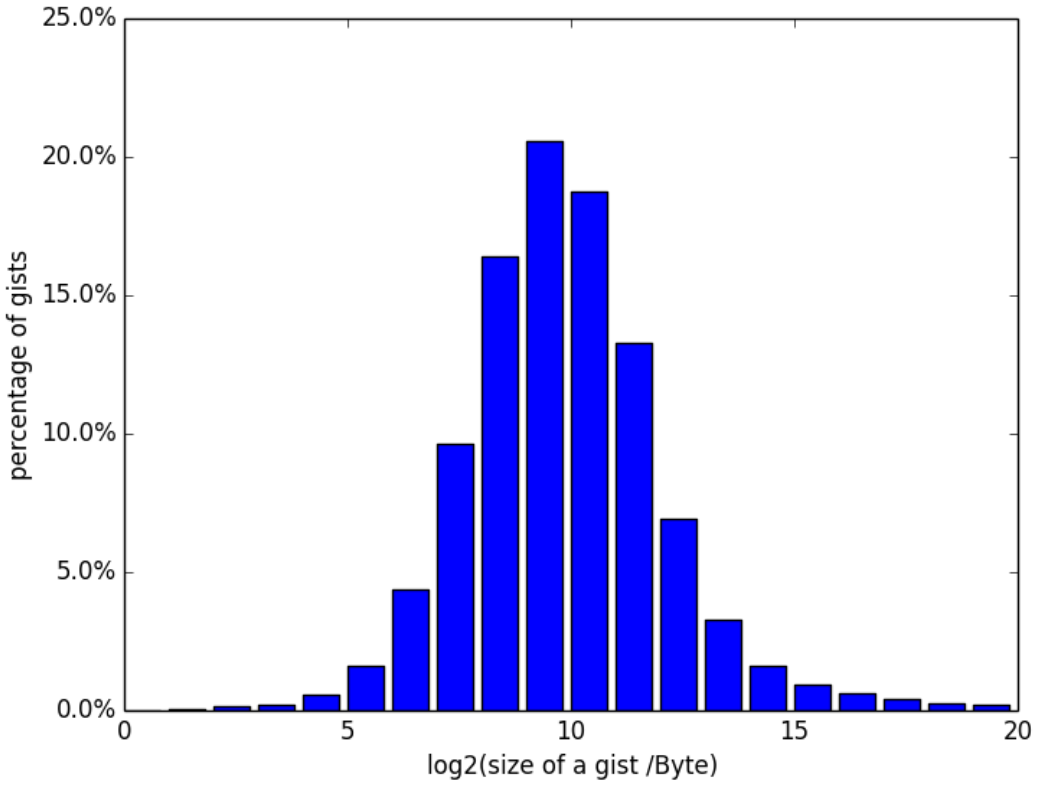

Figure 4.1: Percentage of gists of different size

4.4.2 Lines of code

Since *CLOC* tool would ignore some files automatically[1], we finally got the analysis result of 375,085 gists (60.76% of all obtained gists). The analysis results shown in Figure 4.2, 4.3 and 4.4 illustrate the distribution of gists having different number code/comment/blank lines.

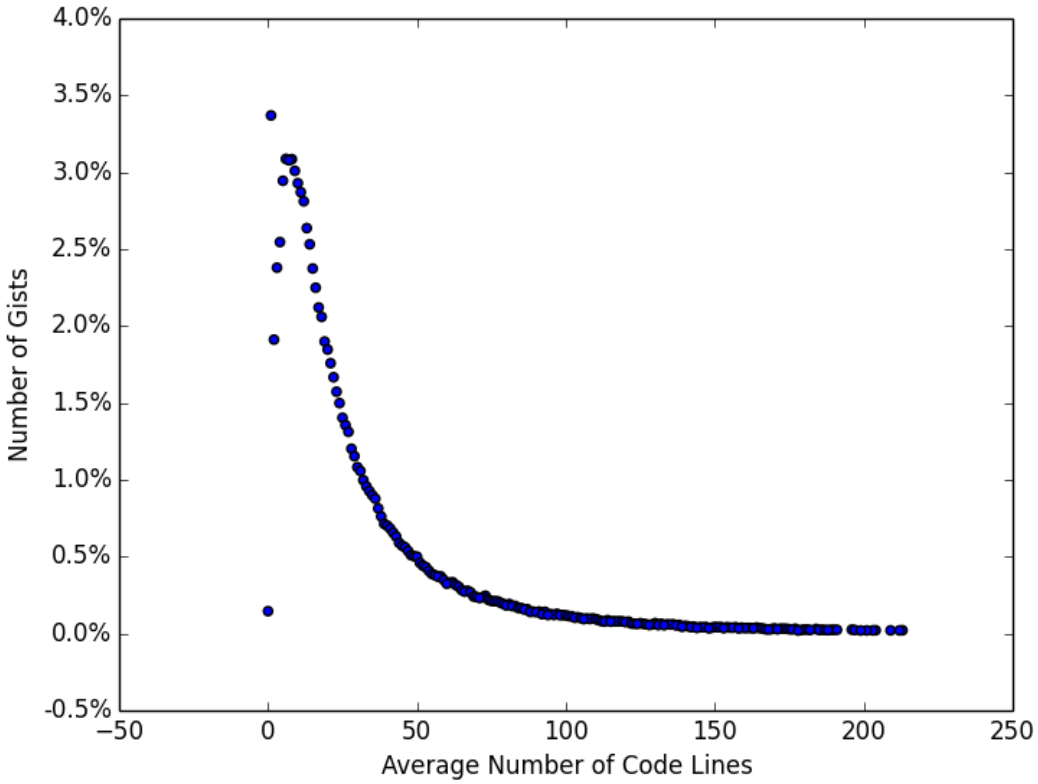

Figure 4.2: Percentage of gists having different number of code lines

Then we also calculated their maximum, minimum, mean, median, mode, standard and variance, as are shown in table 4.6, 4.7 and 4.8. From "average", "median" and "mode", we can see that majority of gists have small number of code/comment/blank lines. From "maximum", we can see there are "outliers" existed, which agrees with the observation from the aspect of "Size". From "variance", we can see a huge difference between circumstances of different files.

[1] *CLOC* will ignore binary and zero-sized files, as well as those not having a recognized extension or not being recognized as a scripting language.

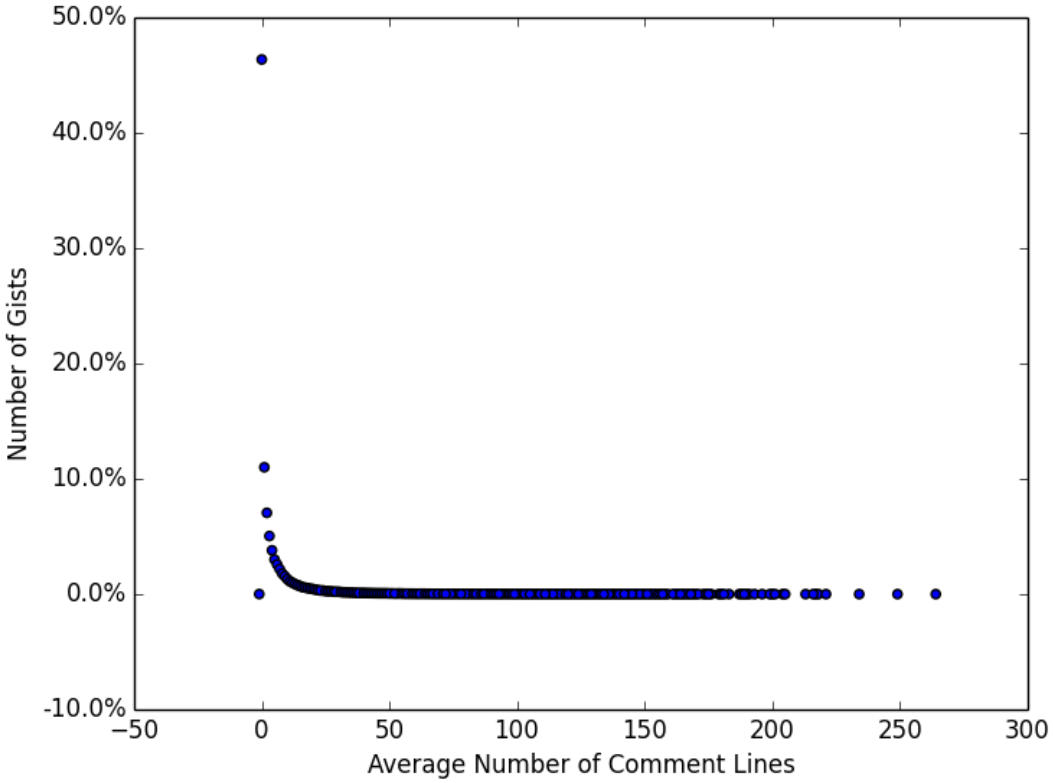

Figure 4.3: Percentage of gists having different number of comment lines

Statistical Metrics	**Value**
Average	58.87
Maximum	77,126
Minimum	0
Median	19
Mode	1
Standard	485.60
Variance	235,807.91

Table 4.6: Statistical metrics for number of code lines in each gist

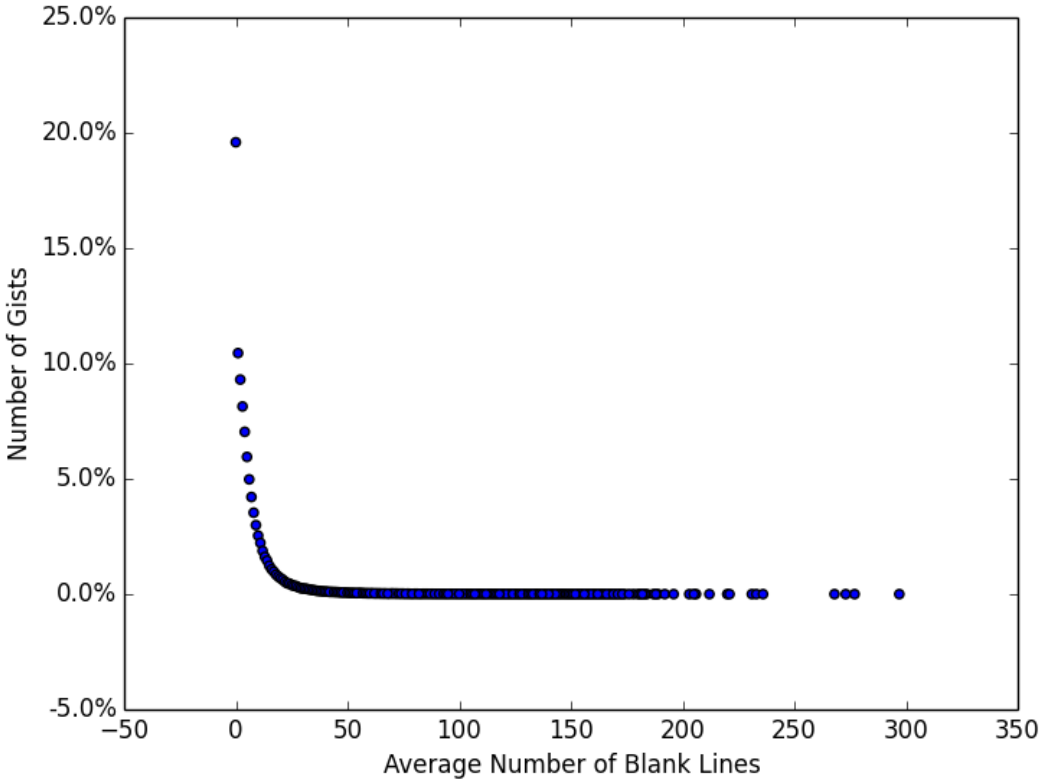

Figure 4.4: Percentage of gists having different number of blank lines

Statistical Metrics	**Value**
Average	8.87
Maximum	13,133
Minimum	0
Median	1
Mode	0
Standard	113.45
Variance	12,870.66

Table 4.7: Statistical metrics for number of comment lines in each gist

Statistical Metrics	**Value**
Average	10.10
Maximum	37,546
Minimum	0
Median	4
Mode	0
Standard	97.46
Variance	9,497.80

Table 4.8: Statistical metrics for number of blank lines in each gist

Then we calculated the number of code/comment/blank lines of gists files in specific languages, and sorted them in descending way by lines of code. Finally we got the result of 91 languages, and the result is shown in table 4.9.

Language	**Average Code Lines**	**Average Comment Lines**	**Average Blank Lines**
OpenEdge ABL	341	19	66
TeX	203	0	22
Smarty	193	48	25
Prolog	191	1	8
YAML	191	3	4
Assembly	187	24	30
Darcs Patch	164	11	14
XML	155	5	5
Pascal	149	12	21
Racket	148	3	20
...	...	...	...
M	18	4	5
Tcsh	16	3	2
Gentoo Eclass	16	0	6
Puppet	14	17	4
Haml	13	1	3
eC	12	0	0
Mathematica	12	0	3
Stata	11	0	4
AsciiDoc	11	0	3
JSON	10	0	1

Table 4.9: Lines statistics for those gists in a specific programming language

4.4.3 Content

400 gists were randomly chosen for manually quanlitative analysis, with each of them being assigned one or several labels based on their content. These gists were labeled in terms of content between files respectively. Labels are demonstrated as follows:

- Code: any kinds of source code;

- Test: test code;
- Class: only a class is defined;
- Template: coding example/pattern;
- Command: commands used in shell;
- Function: only one or several functions are defined;
- Fragment: only one or several lines of code without complete functions/class definitions or implementations;
- Note: script other than source code;
- Log: log files, or error messages after running code;
- Configuration: config files used for build/run code;
- Diff: diff files used for visualizing the changes in a file;
- Documentation: text tutorial documentations;
- Data: data stored in json, csv or other forms;
- Blog: technical blog usually written in markdown files;
- Non-technical: notes without any technical content.

Distribution of labels are shown in table 4.10. We can see that many gists are just source code fragment, functions/class definition, code template, log/configuration files, test code, etc. All these files are usually very short, which also emphasizes our conclusion that most gists are small.

4.5 FINDING 5: Files are always related existing in a gist.

For those 400 gists that were manually analyzed, we also labeled them in terms of relationship between multiple files in a gist. These labels are demonstrated as follows:

- Siblings: files in a gist are independent;

Labels in terms of content	**Percentage of Targeted Gists**
Code	79.25%
Note	24.5%
Function	10.0%
Log	7.5%
Class	6.5%
Data	5.25%
Fragment	4.75%
Command	4.75%
Template	4.0%
Test	3.25%
Configuration	3.25%
Diff	1.75%
Blog	1.5%
Non-technical	1.5%
Documentation	0.5%
Empty	0.5%

Table 4.10: Distribution of gists labeled in terms of content

- Reference: one file refers to another file or calls functions/methods defined in another file;
- Generation: one file is the input/output of another file;
- Test: one file is to test the code of another file;
- Attachment: one file contains the configuration/readme information of another file.

Distribution of labels are shown in table 4.11. We can see that most gists contain only 1 file, for which we assigned label "Single". Other than that, there always exist a relationship between files if there are multiple ones in a gist. These different kinds of relationship are demonstrated as follows:

Relationship between files of each gist	**Percentage of Targeted Gists**
Single	87.25%
Reference	4.25%
Generation	3.25%
Siblings	3.25%
Test	1.25%
Attachment	0.75%

Table 4.11: Distribution of gists labeled in terms of relationship between files

4.6 FINDING 6: Gists are not only source code, but demonstrate a diversity in the content.

Figure 4.5 and table 4.12 show the percentage of gist files using different programming languages. Apart from that, we can also observe that many gists files (27.85%) do not indicate any programming languages, which may contain text/image files or other types of files that are not using a programming language.

4.6.1 Programming languages

Maybe can compare with programming languages used in repos.

Languages Used in Gists	Percentage of Gists Files using this language
None	27.85%
Ruby	9.80%
JavaScript	9.65%
Python	5.91%
Shell	5.62%
Markdown	5.05%
PHP	4.25%
HTML	3.96%
Java	2.46%
JSON	1.93%
CSS	1.84%
C#	1.58%
Diff	1.38%
C	1.32%
XML	1.28%
C++	1.13%
Objective-C	1.06%
Scala	1.04%
CoffeeScript	1.01%
Perl	0.94%
Clojure	0.91%
Haskell	0.72%
SQL	0.62%
YAML	0.50%
Go	0.49%
VimL	0.47%
SCSS	0.40%
Emacs Lisp	0.39%
Groovy	0.33%
R	0.32%
HTML+ERB	0.26%
INI	0.26%
Other Languages	5.24%

Table 4.12: Distribution of gists files using different languages

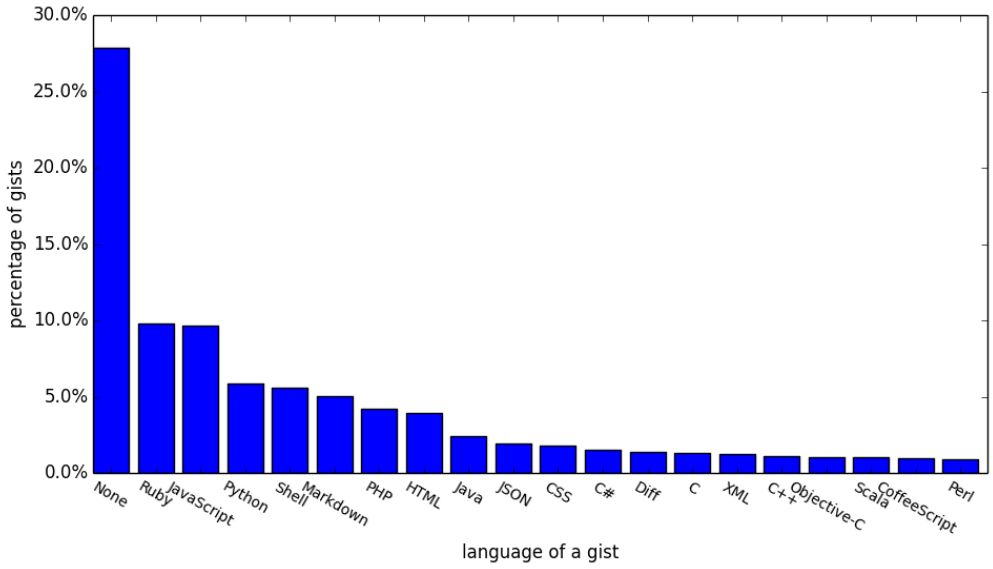

Figure 4.5: Percentage of gists in different languages

4.6.2 None Programming languages

In order to have a better understanding of these gists, more study was conducted for these gists files with "None" language. We looked into the "types" field and calculated the distributions as shown in table 4.13.

file type	percentage
text	95.8825%
image	3.5616%
application	0.5338%
audio	0.0099%
message	0.0072%
model	0.0018%
chemical	0.0014%
video	0.0014%
x-conference	0.0005%

Table 4.13: Types distribution of gists files with no programming languages

We found that majority of "None source code" gists files are text files, along with a less number of image files, application files, audio files, etc, which is really a diversity. In order to know more details of these files, we further studied the file extension of these files. For those files without an extension, we tagged it as "no extension". For each of the types, we found the distribution of different file extensions as the table 4.14, 4.15, 4.16, 4.21, 4.19, 4.17, 4.20, 4.18 and 4.22 show.

file extension	percentage
.ice	100.00%

Table 4.14: Distribution of file extensions for type "x-conference"

What should be noticed is that in type "text", there are 32.31% of files (8.63% of all files obtained) not having a file extension, and for all the files of other types, all files have a file extension. Therefore, we took the advantege of the tool *CLOC* to analyze these "text" files without extensions. Finally, *CLOC* recognized 15,510 (22.69%) files of all those without a file extension, and their distribution is as shown

file extension	percentage
.txt	52.65%
no extension	32.31%
.log	1.84%
.conf	1.79%
.gitignore	0.60%
.ipynb	0.59%
.csv	0.58%
.cfg	0.25%
.geojson	0.23%
.out	0.21%
.config	0.20%
.nix	0.18%
.sublime-snippet	0.15%
.	0.14%
.zsh-theme	0.09%
.screenrc	0.08%
.inc	0.08%
.manifest	0.08%
.tsv	0.07%
.sublime-build	0.07%
Others	7.81%

Table 4.15: Distribution of file extensions for type “text”

file extension	percentage
.png	52.38%
.gif	35.71%
.jpg	10.92%
.jpeg	0.47%
.ico	0.28%
.bmp	0.09%
.pbm	0.06%
.ppm	0.03%
.xcf	0.03%
.pgm	0.01%
.xbm	0.01%
.tga	0.01%
.tiff	0.01%

Table 4.16: Distribution of file extensions for type “image”

file extension	percentage
.xyz	100.00%

Table 4.17: Distribution of file extensions for type “chemical”

file extension	percentage
.class	36.15%
.jar	8.36%
.com	8.02%
.pdf	7.52%
.bin	5.41%
.ttf	3.72%
.gz	3.29%
.zip	2.70%
.apk	2.36%
.exe	1.77%
.dll	1.77%
.woff	1.69%
.dtd	1.69%
.crt	1.60%
.hdf	1.10%
.swf	1.01%
.so	0.84%
.latex	0.76%
.otf	0.68%
.me	0.59%
Others	8.95%

Table 4.18: Distribution of file extensions for type “application”

file extension	percentage
.m4v	66.67%
.mov	33.33%

Table 4.19: Distribution of file extensions for type “video”

file extension	percentage
.eml	100.00%

Table 4.20: Distribution of file extensions for type “message”

file extension	percentage
.mp3	40.91%
.wav	18.18%
.ogg	18.18%
.aif	13.64%
.m4a	4.55%
.rpm	4.55%

Table 4.21: Distribution of file extensions for type “audio”

file extension	percentage
.wrl	100.00%

Table 4.22: Distribution of file extensions for type “model”

in table 4.23. It's clear that these files without a file extension are usually source code files.

Language Recognized	Percentage
Bourne Again Shell	17.5418%
Ruby	16.8471%
Bourne Shell	16.7564%
Javascript	10.1817%
Python	6.1619%
PHP	4.2237%
Perl	2.9979%
Java	2.6279%
HTML	2.3132%
C#	1.6588%
CoffeeScript	1.2787%
C	1.2057%
ASP.Net	1.1050%
Scala	0.9766%
CSS	0.9590%
XML	0.9339%
C++	0.8911%
Lisp	0.8306%
Haskell	0.7300%
Others	9.7790%

Table 4.23: Language distribution of those gists files without an extension

In summary, most of gists are source code files, but there still exists many gists containing various other kinds of files, including application files, text files, audio files, video files, etc.

4.7 FINDING 7: Majority gists have never been updated or once.

Figures 4.6 and Tables 4.24 show the percentage of gists that have different number of commits since they were created. We can see majority of gists have been commited once or twice. Given that the creation of a gist is a commit, we can infer that people barely update their or other people's gists. However, there also exist gists that have a very large number of commits.

Number of Commits One Gist Has	Percentage of Gists
1	62.95%
2	18.35%
3	7.72%
4	3.85%
5	2.15%
6	1.30%
7	0.84%
8	0.57%
9	0.43%
10	0.32%
11	0.24%
12	0.19%
≥ 13	1.09%

Table 4.24: Distribution of gists having different number of commits

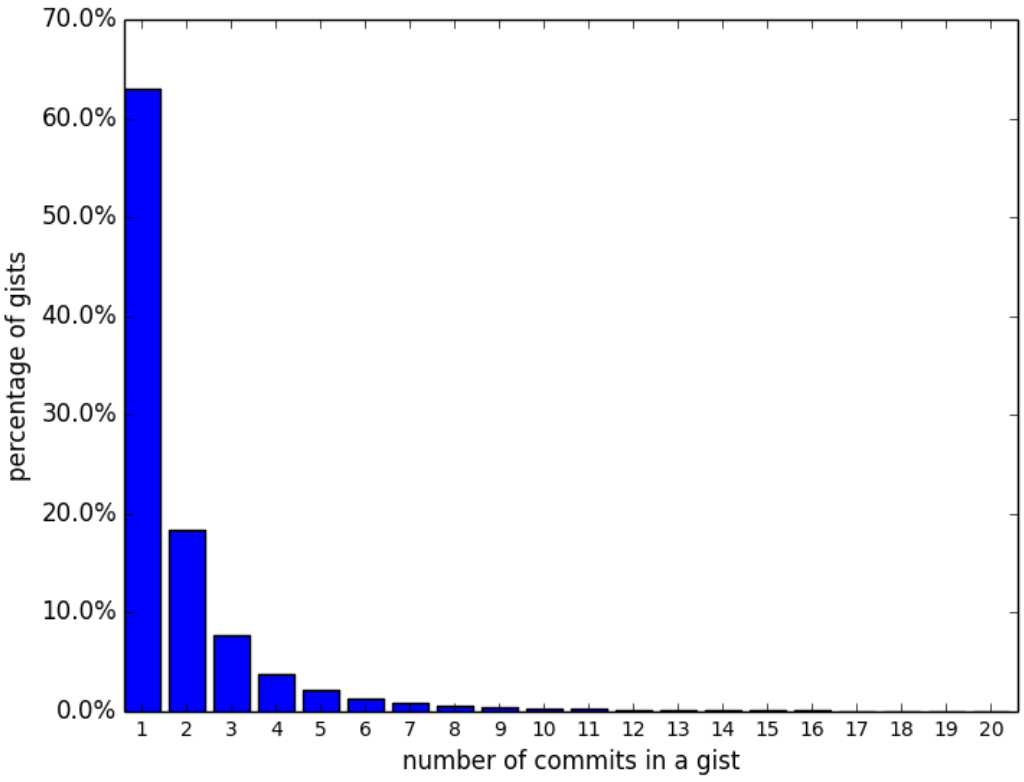

Figure 4.6: Percentage of gists having different commits

Another evidence of this finding can be obtained from the "created at" and "updated at" information in the metadata, which mean the date of being created and the date of being updated most recently. The difference between these two dates for all the obtained is as figure 4.7 shows.

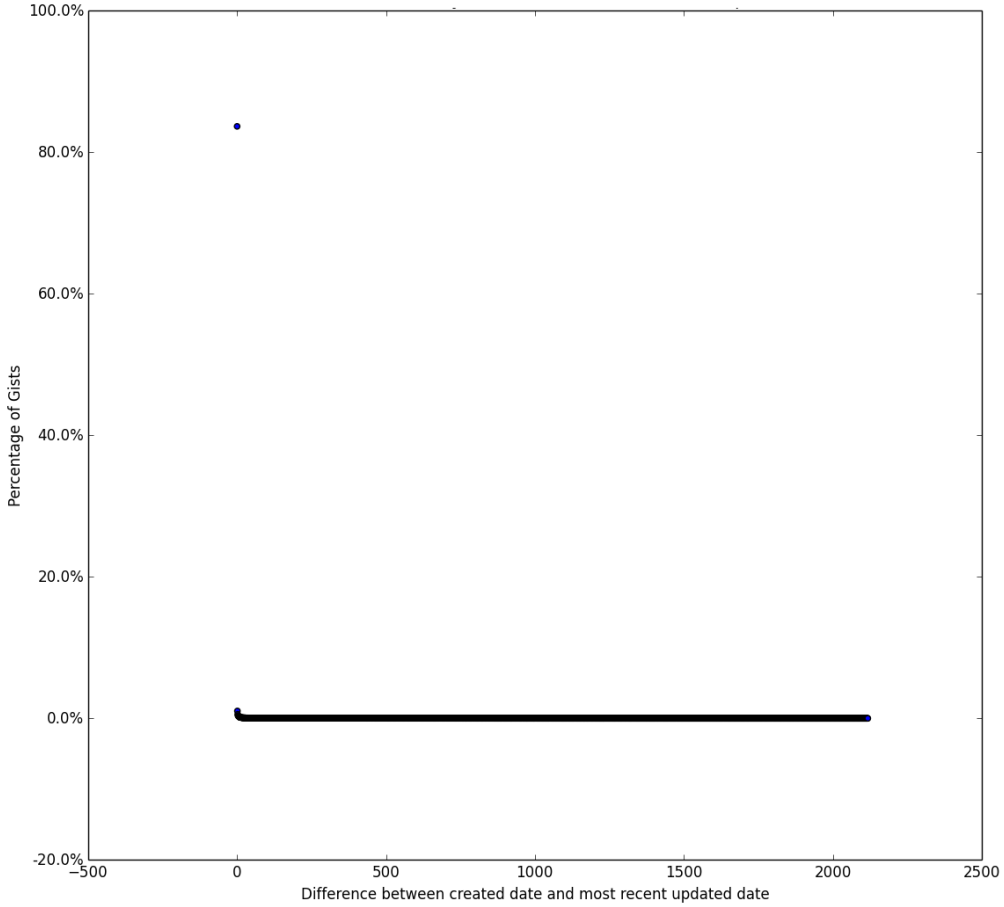

Figure 4.7: Gists Distribution with Days Between Created Date and Updated Date

4.8 FINDING 8: Majority gists have never been commented.

Figures 4.8 and 4.25 show the percentage of gists that have different number of comments. We can see majority of gists have no comments, from which we can infer that people barely discuss gists. Instead, they usually commit to their own gists. However, there also exist gists that have a very large number of comments.

Number of Comments One Gist Has	Percentage of Gists
0	92.67%
1	4.56%
2	1.26%
≥ 3	1.51%

Table 4.25: Distribution of gists having different number of comments

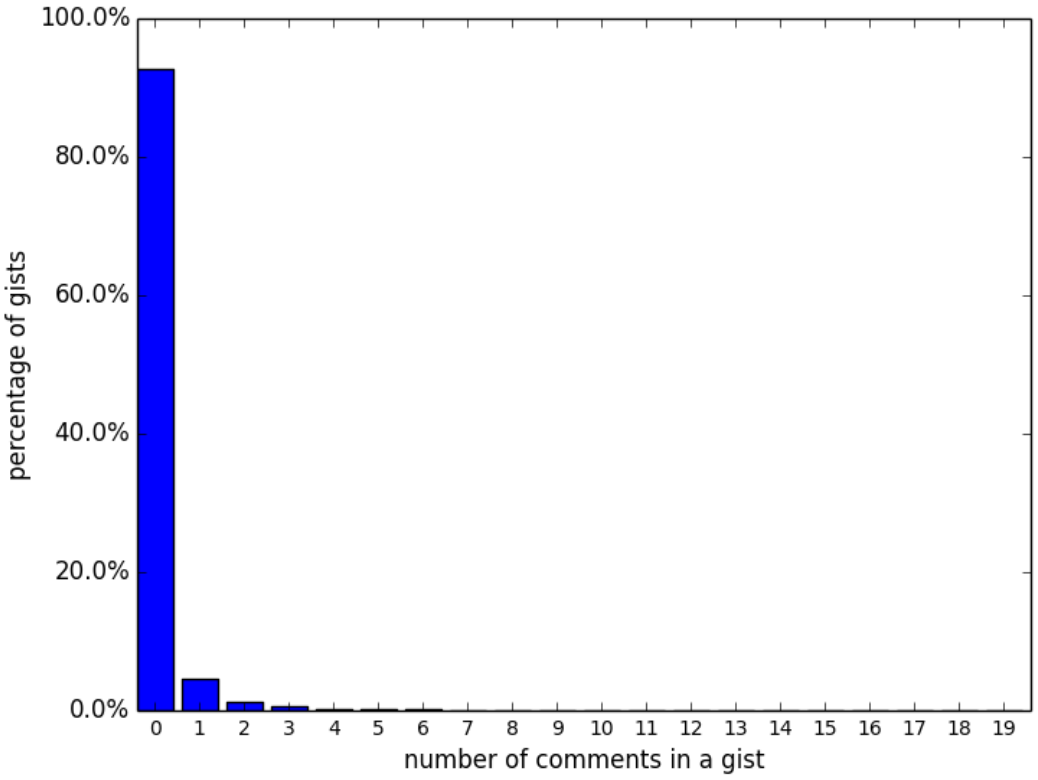

Figure 4.8: Percentage of gists having different comments

4.9 FINDING 9: Majority gists have never been forked.

Figures 4.9 and Tables 4.26 show the percentage of gists that have different number of forks. We can see that majority of gists have never been forked. We can infer that people barely collaborate on gists, which means they only use gists for personal purpose. However, there also exist gists that have a very large number of forks, which we will do further research later.

Number of Forks Each Gist Has	Percentage of Gists
0	94.58%
1	3.98%
≥ 2	1.44%

Table 4.26: Distribution of gists having different number of forks

4.10 FINDING 10: People use gists in many ways.

We used "github gist" as keywords to search for related tweets, and read all the obtained tweets one by one. While reading these data, we tried to classify them into several categories of different ways of using gists. Our results are demonstrated as follows:

- Bug Tracking

  Some people may track software bugs and record their finding in GitHub gists.

  "Yeah, I love Gists, the Issue tracker, the code viewer/editor, etc. Very professional. #GitHub" [2]

- Being embeded in blogging platforms.

  One of the biggest blog websites, WordPress[3] has supported embedding gists in the blogs[4]. Apart from that, Many other blogging platforms like Medium[5] also come to support such embedding.[6]

[2] https://twitter.com/pcgeek86/status/484328683839045632
[3] https://wordpress.com/
[4] http://crunchify.com/how-to-embed-and-share-github-gists-on-your-wordpress-blog
[5] https://medium.com/
[6] https://medium.com/the-story/yes-we-get-the-gist-1c2a27cdfc22

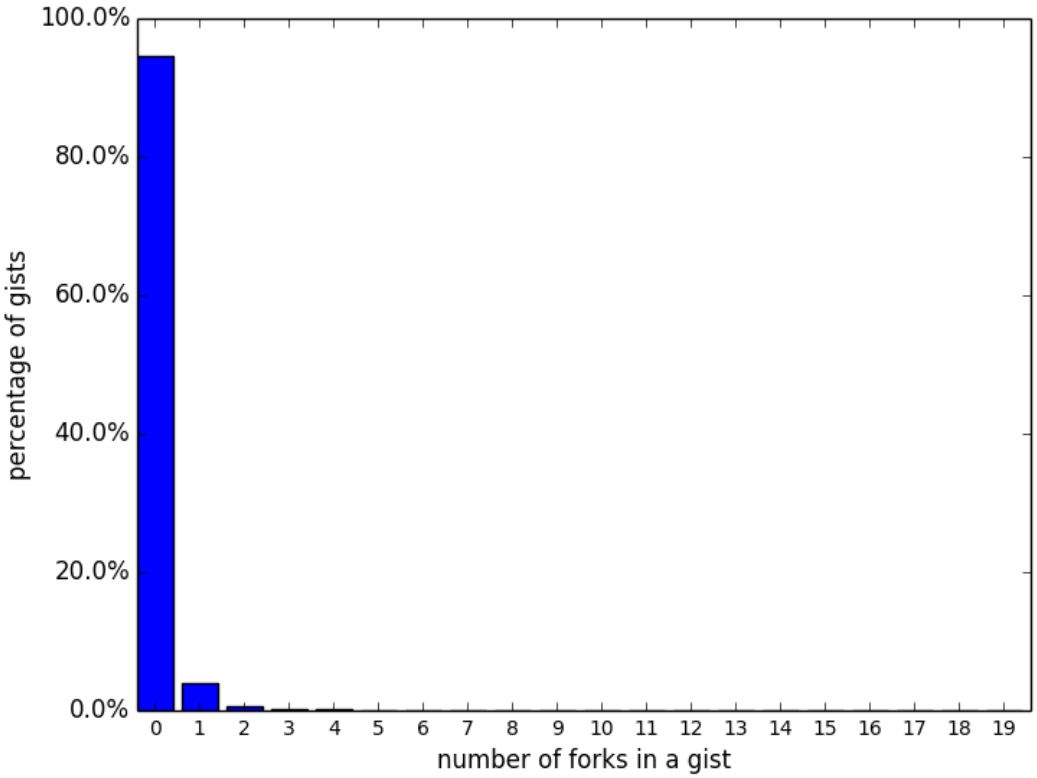

Figure 4.9: Percentage of gists having different forks

"I think I might start using GitHub Gists in my blog posts. Thinking about how to integrate it smoothly." –by one Twitter user.[7]

- Version-controlled list

  Some users come up with some non-trival ways to make full use of gists. One good example is a popular blog being widely tweeted on Twitter, which is to teach people how to maitain a to-do list using a private github gist authored by Carl Sednaoui[8]

  "GitHub Gists are a great way to keep version-controlled lists (in this case, US states I've visited) https://gist.github.com/dliggat/11003570..."[9]

- Saving learning outcome

  Gists can also be used to take notes that are usually about learning outcomes. For example, after a discussion about a technical problem through Twitter comments, a user learned something from it and he said,

  "with more help from @qwzybug just got Drift successfully prompting for filenames in a sheet and posting gists: http://gist.github.com/127815"[10]

  More examples are as follows:

  "I've been saving my learning as gists because blogging is a barrier: https://gist.github.com/Greg Boggs"[11]

  "Useful fiddles and gists collected from #AngularJS forum discussions https://github.com/angu Examples..."[12]

- Files sharing

  GitHub gists could also serve as a common pool of shared files, such that when a person wants to show a file to others, he just needs to send the gist link to his partners instead of the whole file. The evidence is as the conversations show below.

  A: "Can you show me what your application.xml looks like for your ios+android game? Need to know what info is needed to compile."

[7] https://twitter.com/BenNadel/status/157495925231714304
[8] http://carlsednaoui.com/post/70299468325/the-best-to-do-list-a-private-gist
[9] https://twitter.com/dliggat/status/458090816930848768
[10] https://twitter.com/atduskgreg/status/2115301909
[11] https://twitter.com/gregory_boggs/status/483111455550877697
[12] https://twitter.com/AppsHybrid/status/481901294563901440

B: "I can put it on github gists sometime soon, remind me in a few days if I forget..." [13]

A: "Can ChefSpec load data bags from a directory via Rspec config, similar to role_path?" B: "it is, but I still need to make it bettereerer. Here's a quick example: https://gist.github.com/..." [14]

- Collaboration

  Gists also act as a tool to help people with collaboration. People can put their work in a private gist, and every member can commit to it, as is inferred from the following tweets.

  "Every once in a while I think I wish I could "draft" a Pull-request, issue, or comment on GitHub, then I remember that private gists exist." [15]

  "work so far is at https://gist.github.com/..."[16]

[13] `https://twitter.com/stvr_tweets/status/483092598639185920`
[14] `https://twitter.com/mikefiedler/status/482271582820126720`
[15] `https://twitter.com/nuclearsandwich/status/249213040610910209`
[16] `https://twitter.com/danbri/status/482152387445288960`

Chapter 5

Limitations and Future Work

When counting the lines of code, the tool *CLOC* was used. However, this tool has some limitations[1]. For example, it treats lines containing both source code and comments as lines of code; comment markers within strings are treated as actual comment markers and not string literals; Lua long comments could not be recognized. Although we can see a general pattern from the analysis results, these limitations would always affect the statistical precision to some degree. Thus a better source code analysis tool is needed.

[1] `http://cloc.sourceforge.net/#Limitations`

Chapter 6

Conclusion

Bibliography

[1] Scott Chacon and Junio C Hamano. *Pro git*, volume 288. Springer, 2009.

[2] B. de Alwis and J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on*, pages 36–39, May 2009.

[3] P.C. Rigby, E.T. Barr, C. Bird, P. Devanbu, and D.M. German. What effect does distributed version control have on oss project organization? In *Release Engineering (RELENG), 2013 1st International Workshop on*, pages 29–32, May 2013.

[4] D. Spinellis. Git. *Software, IEEE*, 29(3):100–101, May 2012.

[5] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Leveraging transparency. *Software, IEEE*, 30(1):37–43, Jan 2013.

[6] A. Capiluppi, A. Serebrenik, and L. Singer. **Assessing Technical Candidates on the Social Web**. *Software, IEEE*, 30(1):45–51, Jan 2013.

[7] Jennifer Marlow and Laura Dabbish. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 145–156, New York, NY, USA, 2013. ACM.

[8] Rahul Venkataramani, Atul Gupta, Allahbaksh Asadullah, Basavaraju Muddu, and Vasudev Bhat. Discovery of technical expertise from open source code repositories. In *Proceedings of the 22Nd International Conference on World Wide Web Companion*, WWW '13 Companion, pages 97–98, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[9] C. Teyton, J.-R. Falleri, F. Morandat, and X. Blanc. Find your library experts. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 202–211, Oct 2013.

[10] T.F. Bissyande, F. Thung, D. Lo, Lingxiao Jiang, and L. Reveillere. Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pages 303–312, July 2013.

[11] Anirban Majumder, Samik Datta, and K.V.M. Naidu. Capacitated team formation problem on social networks. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 1005–1013, New York, NY, USA, 2012. ACM.

[12] Jennifer Marlow, Laura Dabbish, and Jim Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in github. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 117–128, New York, NY, USA, 2013. ACM.

[13] Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Treude, Margaret-Anne Storey, and Kurt Schneider. Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 103–116, New York, NY, USA, 2013. ACM.

[14] Joohee Choi, Junghong Choi, Jae Yun Moon, Jungpil Hahn, and Jinwoo Kim. Herding in open source software development: An exploratory study. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work Companion*, CSCW '13, pages 129–134, New York, NY, USA, 2013. ACM.

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

[16] G. Gousios and D. Spinellis. Ghtorrent: Github's data from a firehose. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 12–21, 2012.