Study the Correlation Between the readme File of GitHub Projects and Their Popularity

Tianlei Wang[a], Shaowei Wang[a], Tse-Hsun (Peter) Chen[b]

[a]*Department of Computer Science, University of Manitoba, Canada*
[b]*Department of Computer Science and Software Engineering, Concordia University, Canada, Canada*

Abstract

A readme file plays an important role in a GitHub repository to provide a starting point for developers to reuse and make contributions. A good readme could provide sufficient information for users to learn and start a GitHub repository and might be correlated to the popularity of a repository. Given the importance of the role that a readme file plays, we aim to study to understand the correlation between the readme file of GitHub repositories and their popularity. We analyze readme files of 5,000 GitHub repositories across more than 20 languages. We study the relationship between readme file related factors and the popularity of GitHub repositories. We observe that: 1) Most of the studied readme file related factors (e.g., the number of lists, the number and frequency of updates on the readme file) are statistically significantly different between popular and non-popular repositories with non-negligible effect size. 2) After controlling repository-specific factors (e.g., repository topics and license information), the number of lists and the frequency of updates are the most significantly important factors that discriminate between popular and non-popular repositories. 3) The most of updates were made to update references in popular repositories, while in non-popular repositories most updates are for the content of how to use the repository.

Keywords:
Documentation, Github, readme file, random forest

Email addresses: wangt316@myumanitoba.ca (Tianlei Wang), shaowei.wang@umanitoba.ca (Shaowei Wang), peterc@encs.concordia.ca (Tse-Hsun (Peter) Chen)

1. Introduction

GitHub, the largest open-source projects hosting site in the world, provides a platform for developers and organizations to share and host their open-source projects. GitHub hosts more than 200 million repositories, as of March, 2022 (GitHub, 2022c). Millions of developers and organizations contribute to these repositories by adding new features and resolving bugs (Bao *et al.*, 2019; McDonald and Goggins, 2013; Jiang *et al.*, 2017). Developers also learn from the hosted projects on GitHub and utilize them in their projects (Gharehyazie *et al.*, 2019, 2017).

A readme file plays an important role in a repository to provide a starting point for developers to understand and learn the repository. As the official documentation supported by GitHub, a readme file typically provides the basic information related to the repository, such as what the project does, how users can get started with the project, etc (GitHub, 2022a). The information provided in the readme file largely affects the contribution and the usage of a repository. A well-organized and maintained readme could provide sufficient information for users and reduce the learning curve of getting started on the project for users (Steinmacher *et al.*, 2014), which probably correlates to the popularity of the project. For instance, a repository with a well-maintained readme file, in which all information updated to date, may be easy for a user to get started on it, compared with a repository that has a significant amount of out-of-dated information in the readme file, as a result, attracts more developers to use it and contribute to it. Therefore, we aim to understand the correlation between various factors related to readme file of a GitHub repository and the popularity of the repository, e.g., whether the organization and update activities of a readme file correlate to the popularity of a repository. Notice that our aim is not to uncover causal relationships between various factors and the popularity of GitHub repositories. Instead, we compare the differences between popular and non-popular GitHub repositories, aiming to help highlight good practices in improving readme files.

In this study, we performed an analysis of the readme files of 5,000 GitHub repositories across different domains and languages. We study the relationship between readme file related factors and the popularity of GitHub repositories to understand whether the factors related to readme files are different between popular and non-popular GitHub repositories, and which factors are

most important for distinguishing them. For this purpose, we structure our inquiry using 18 factors along three dimensions: the content presented and organized in the readme file (e.g., the number of lists, images, and links), updates on the readme file (e.g., the number and frequency of updates), and repository meta information (e.g., topics, age, license, and size). More specifically, we structure our study along the following three research questions:

- **RQ1: Is there a relationship between the studied readme file related factors and the popularity of GitHub repositories?**
  In RQ1, we compare the readme file related factors between popular and non-popular repositories and examine if the differences between those two groups are statistically significant. We observe that all studied readme file related factors (e.g., number of links, images, and updates frequency) except heading content are statistically significantly different between popular and non-popular repositories with non-negligible effect size. Popular and non-popular repositories share similar headings (e.g., usage and installation).

- **RQ2: What are the most important readme file related factors that differentiate popular and unpopular GitHub repositories?**
  In RQ2, we build a classification model to investigate which readme file related factors are the most important for discriminating between popular and non-popular repositories by controlling factors related to the repository. We observe that after controlling factors related to the repository (e.g., repository topics and license information), the number of lists, links, and the frequency of updates are statistically significantly important factors that discriminate between popular and non-popular repositories. The frequency/number of readme updates and the number of lists and links positively correlate with the likelihood of a repository being popular.

- **RQ3: What content is updated in readme files between popular and non-popular GitHub repositories?**
  For RQ3, we perform qualitative analysis to understand what content is updated in readme file of popular and non-popular repositories. We observe that the largest portion of the updates (47%) were performed

to update references in popular repositories, while the largest portion of updates were performed to update the content of "how" in non-popular ones (34%). A remarkable portion of updates on readme files were performed to improve the presentation and update badges for both popular and non-popular repositories.

In summary, the quality of the readme file (e.g., proper organization of content and ensuring content is up to date) is positively correlated with the popularity of repositories after controlling for repository-specific factors. Developers should pay attention to improving the readme file, e.g., improving the organization of content and ensuring the content is up to date. Future research is encouraged to develop approach to help maintain readme automatically (e.g., detecting and updating broken/obsolete references) and generate readme file templates automatically. To enrich future research on this direct, we make our replication publicly available `https://github.com/TianleiWang0414/WIP2022-GitHub_Readme-code/`.

Paper Organization. Section 2 introduces the background information about the GitHub readme file and related work. Section 3 describes our data collection process and studied factors. Section 4 presents the result of our research questions. Section 5 describes the discussion on update activities of readme files, implications of our findings, and the threats to the validity of our observation. Finally, Section 6 concludes the paper.

2. Background & Related work

2.1. Readme file on GitHub

Readme files are text files, typically written in markdown, that provide an overview of information to allow an individual to understand how to run, when to use, and what tools to apply for a given project. Usually, readme files are the first documentation/file of a project that is visited. Readme files usually follow a strict naming convention, as the name suggests, they are often written as "readme.md". Suggested by GitHub official documentation, readme files typically contain the following information: 1) What the project does; 2) Why the project is useful; 3) How users can get started with the project; 4) Where users can get help with your project; 5) Who maintains and contributes to the project (GitHub, 2022a). Readme files are part of the GitHub repositories and they can be updated accordingly as the projects

evolve. Figure 1 shows an example readme file of a GitHub repository[1]. Various pieces of information are labeled in the example, such as heading, image, badge, and link.

2.2. Related work

2.2.1. Studying and leveraging the content presented in readme files

Readme files are considered an important source for understanding repositories not only by developers but also by researchers. Therefore, there has been a number of studies that leverage readme files to conduct software engineering tasks (e.g., project recommendation and project categorization). Zhang *et al.* (2017) proposed an approach that leverages the content of readme files together with their sources (e.g., source code), to detect similar repositories on GitHub. They assume that similar repositories have similar readme files in terms of content. Similarly, Koskela *et al.* (2018) leveraged the content presented in readme files together with the tags of repositories to recommend GitHub repositories for users based on their profiles. Sharma *et al.* (2017) categorized GitHub projects based on the content in readme files. Coelho *et al.* (2018) leverage the information presented in readme files to identify unmaintained GitHub projects. Trockman *et al.* (2018) conducted an empirical study to understand the repository badges in the npm Ecosystem shown on the readme files. Ikeda *et al.* (2019) investigate the content of readme file for JavaScript packages and its relationship with the type of a package. Different from prior studies that focus on leveraging readme file as a source to perform other tasks, our study focuses on studying the readme file itself and investigating its association with the popularity of GitHub repositories. Hassan and Wang (2017) proposed technique to automatically extract software build commands from software readme files and Wiki pages for software building.

Several studies have been done to investigate the content of readme files of GitHub repositories. Liu *et al.* (2022) study the patterns of readme files and determine the degree to which readme files are aligned with the official guidelines. They found that the majority of readme files do not align with the GitHub guidelines and repositories whose readme files follow the GitHub guidelines tend to receive more stars. Prana *et al.* (2019) studied the content of 393 GitHub readme files. Their study shows that the information about

[1]https://github.com/alibaba/Sentinel

Figure 1: Example readme file of a GitHub repository.

"what the project does" and "how users can get started with the project" of a repository is common while information on the purpose and status is rare. They developed a classifier to predict categories of sections in the readme files with an F1 score of 0.746. Ikeda *et al.* (2019) studied the readme file of JavaScript packages, and they show that information related to usage, installation, and license are commonly contained by such readme files. Treude *et al.* (2020) recruited technical editors to assess various software documentation, including readme files for R GitHub Projects and Stack Overflow threads in terms of ten dimensions such as readability, cohesion, and structure. They shed light on challenges for software documentation and directions for future studies. Different from these studies that focus on studying the content of readme files, our work aims to investigate the relationship between readme file related features of GitHub projects and their popularity.

2.2.2. Studying popularity of GitHub repositories

Many studies have analyzed software code repositories to understand what makes a code repository popular. Weber and Luo (2014) proposed a classifier using 38 features in various dimensions (e.g., code, project) to classify popular and non-popular Python GitHub projects. Zhu *et al.* (2014) studied the patterns of folders used by 140k Github projects and their relationship with project popularity. They find that the standard folders, such as documents, testing, and examples, are not only among the most frequently used, but their presence in a project is associated with increased chances that a project's code will be forked. Aggarwal *et al.* (2014) investigated the correlation between the documentation change of 90 GitHub projects and their popularity. They observed that the consistent popularity of projects attracts consistent work on documentation. Different from their study, we not only investigate the features related to the revision of readme files, but also other features in terms of structure and content of the readme files. Borges *et al.* (2016) identified four patterns of popularity growth of projects on GitHub in terms of stars. They observed that the main factors that impact the number of stars of GitHub projects are programming language and the application domain. We consider programming language and application domain (we consider tags of a repository) as controlling factors in this study. Different from previous studies, we focus on examining the relationship between features related to readme files and Github projects' popularity and controlling several repository related factors, such as programming language, license, topics, size, and age.

The most related work is a study by Fan *et al.* (2021), which studied the correlation between features from various dimensions (i.e., code, reproducibility, and documentation) of AI repositories and their popularity. Their results show that popular and non-popular AI repositories are significantly different in various features, such as the number of links, images in the README file, and license. Different from Fan et al's study which focuses on investigating academic AI repositories, our study focus on more general GitHub repositories.

3. Data Preparation

3.1. Repository Data Collection

To have enough readme files to answer our RQs, we randomly sampled 5,000 GitHub repositories from Google big query as of January, 2022 (Google, 2020). We retrieved the owner information as well as the repository URL from the Google big query. We then use GitHub API (GitHub, 2022b) to retrieve the README.md files in the main/master branch for the sampled 5,000 repositories. The studied repositories host projects that span more than 20 programming languages including popular ones, such as Python, Java, JavaScript, Go, and C++.

To ensure the quality of our studied repositories, we future filter repositories based on certain criteria. We first removed the repositories that do not have a readme file (i.e., "README.MD"). We filtered out 132 such repositories. We removed the repositories that have default content in the readme file. GitHub initializes the content of a readme file with the repository name if the creator does not modify the readme file. The reason why we removed such repositories is that they do not contain any useful information for our RQs (only contain the name of the repositories). We removed the repositories whose readme file is not fully written in English. We removed such repositories since it requires translation of the whole file to extract useful information, since translation may introduce bias. We used a library called *langdetect* [2] to detect the language written in the readme files. We further removed 614 repositories during the above process. GitHub hosts repositories (academic/tutorial repositories) other than software repositories that could introduce bias (Munaiah *et al.*, 2017; Liu *et al.*, 2022). We removed

[2]https://pypi.org/project/langdetect/

the repositories that have less than 50% of the files that are not written in programming languages and have only one contributor. Last, we removed the repositories that have less than one year from their creation date to our collected date. We did so to ensure the repositories have enough time to attract attention from the community and obtain starts. We ended up with 1,566 repositories.

3.2. Data Labeling

To conduct our analysis, we need to categorize repositories as popular or unpopular. For this purpose, we collected repositories' metadata including stars, watches, number of pull requests, number of contributors, and forks via Github API to find what can be represented for popularity. With this information, we applied Spearman's rank correlation to them. Figure 3 shows that there is a high correlation between forks, watches, and stars. Thus, we believe it is appropriate to select stars as the proxy for popularity following prior studies (Han *et al.*, 2019; Borges *et al.*, 2016). We find that the number of stars is highly skewed and only a small portion of repositories received many stars (see Figure 2). We label the repositories that have a star of 0 as non-popular and the ones that have a star larger than 100 as popular. Note that we use a clear-cut since we would like to mitigate the bias from the repositories with a very similar number of starts while belonging to two different categories and provide a clear boundary between popular and non-popular repositories. For instance, if we considered a threshold of 50 as the boundary to categorize the repositories with at least 50 stars as popular repositories and the ones with less than 50 as non-popular repositories, then a repository with 49 stars and a repository with 51 would be considered as different classes although their stars are very close. Nevertheless, such a clear-cut threshold probably would introduce bias to our study and cause a threat to the validity of our study. We would discuss it in more detail in Section 5.3. We collected 810 non-popular repositories and 756 popular ones. Table 1 shows the information of two groups of repositories.

3.3. Factors Collection

In RQ1, we compare the factors related to the readme file between popular and non-popular repositories (see Section 4.1 for details). In RQ2, we build a classification model to understand the most important readme file related factorsdiscriminating between popular and non-popular repositories by controlling factors that are related to their corresponding repositories.

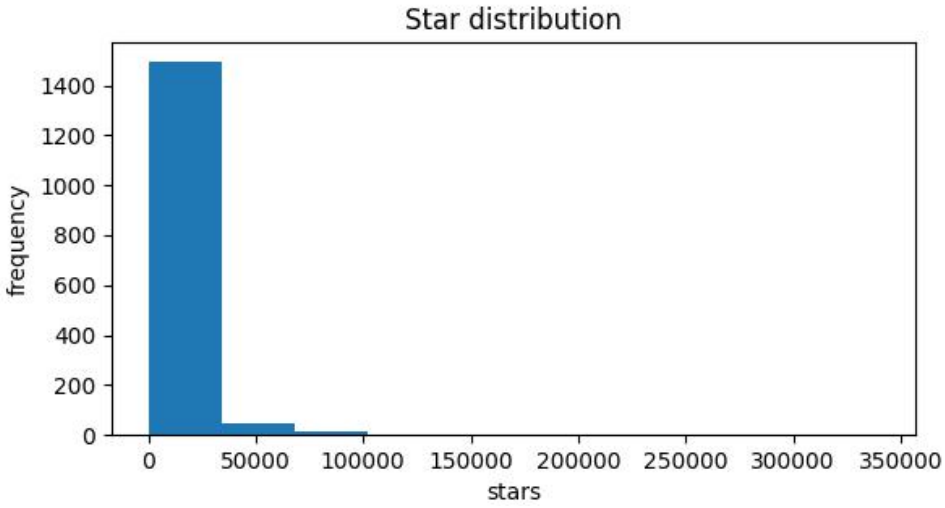

Figure 2: Distribution of repositories with different numbers of stars.

Factor	**Popular**	**Non-popular**
Star	13,398.2/4,374/677/339,873	0/0/0/0
Fork	3,029.7/939/25/86,266	0.25/0/0/6
Watch	516.8/202/7/8,472	2.4/2/1/57
Size (in KB)	137,704.4/14764.0/0/22,309,053	18,339.4/603/0/1,222,458
Age (in day)	2,280.8/2,137/770/4,808	2,446.8/2,420/77/3,832

Table 1: Statistics of popular and non-popular repositories. The value in each cell is ordered as mean/median/min/max.

Table 2: Studied factors that are related to three dimensions readme content, readme update, and repository meta.

	Factor Name	Explanation (data type)	Rationale
Readme content	*R_NumCodeBlock*	The number of code blocks in the readme file (numeric).	Good presentation of readme file of a GitHub Repositories probably could provide a good impression to visitors and improve its popularity. For instance, the list is recommended for clear writing (Kimble, 1992; Fan *et al.*, 2021).
	R_NumIndent	The number of indents in the readme file (numeric).	
	R_NumList	The number of lists in the readme file (numeric).	
	R_NumLink	The number of links in the readme file (numeric).	Providing links and images to further documentation and support channels could help visitors to get started and improves the popularity of the repository.
	R_NumImage	The number of images in the readme file (numeric).	
	R_NumBadge	The number of badges in the readme file (numeric).	The badge indicates the quality and properties of a repository and may have a correlation with the popularity of the repository.
	R_heading	The words appear in headings of the readme file (text).	The popularity of a repository may correlate with the content presented in the readme file. We use the content in the headings as the proxy to represent the major topics presented in the readme file.
Readme update	*R_UpdateInterval*	The average update interval between updates (calculated as *total elapsed days since creation / number of updates)* (numeric).	Frequently update on documentation of a project have positive correlation with its popularity (Aggarwal *et al.*, 2014).
	R_NumUpdate	Number of updates for the readme file (numeric).	
	R_SinceLastUpdate	Number of days since last update in the readme file (numeric).	The time of being inactive of readme file of a project may have negative correlation with its popularity.
Repository meta	*P_Age*	Age of a repository since its creation in days (numeric).	An older repository tends to get more attention from the community and probably has more stars, forks, etc.
	P_size	The size of a repository in KB. (numeric).	Larger size of a repository might indicate more features or better functionality, which might be correlated with its popularity (Tian *et al.*, 2015).
	P_-Max/Min/Avg/Median_-Tag	The max, min, average, and median value of tag popularity scores for a repository. Tag popularity score is calculated as the number of repositories in GitHub that have a tag (numeric).	The tags of a repository indicate its application domain and related topics. The repositories with popular tags may get more attention from the community and probably have more stars, forks, etc (Zhou *et al.*, 2020a; Borges *et al.*, 2016).
	P_License	Type of license the repository has (categorical).	Users can clearly understand the license of a repository and facilitates its popularity (Fan *et al.*, 2021; Laurent, 2004).
	P_ProgramLanguage	The majority of programming languages used in the repository (categorical).	Prior study by Borges *et al.* (2016) observed that the one of main factors that impact the number of stars of GitHub projects is the programming language.

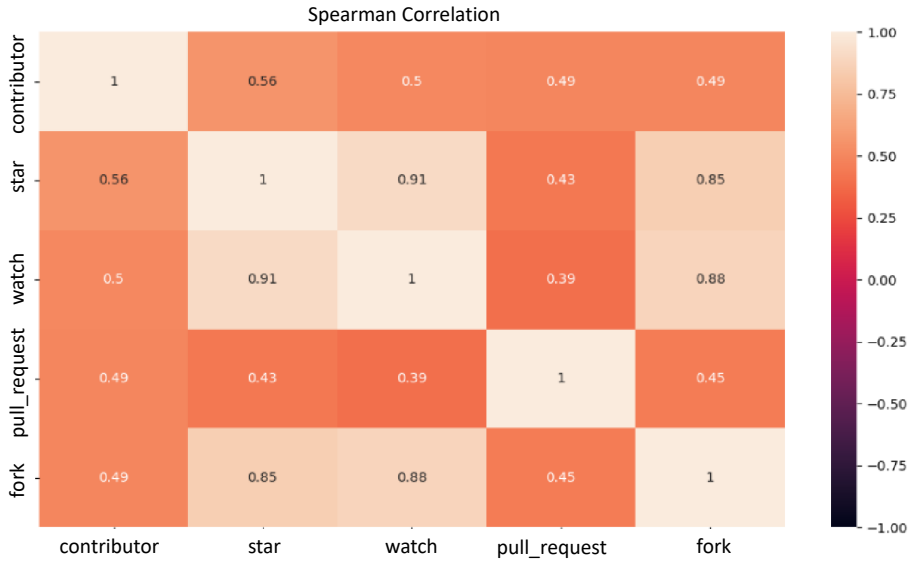

Figure 3: Correlation matrix between the number of forks, pull requests, watches, stars, contributors. We observe that fork, watch, and star have a high correlation.

In this subsection, we discuss the studied factors and how we collect them. The quality of a readme file probably affect the popularity of the corresponding repository. A well-maintained and well-organized could provide sufficient information for users and reduce the learning curve of getting started on the project for users. We collect the factors related to readme file's quality from two dimensions: readme content and readme update. Readme content and update consider the quality of the readme file from static and dynamic perspectives, respectively. The content dimension reflects if a readme file is well-organized, and the update dimension reflects if a readme file is well-maintained. In terms of content, we consider the factors that are captured at the moment of data collection. For content dimension, we first consider the factors that characterize the presentation of a readme file (i.e., *R_NumCodeBlock*, *R_NumImage*, *R_NumIndent*, and *R_NumList*). We expect that a readme file with a good presentation provides a good impression to visitors and correlates with the popularity of the repository. GitHub users usually use markdown for organizing their readme files. Therefore, we consider the formatting types that are commonly used in markdown. For instance, we collect the number of code blocks, indents, lists, and images. Kimble (1992) recommends using lists for clear writing. Tian *et al.* (2015) show the images

displayed in a mobile app have a positive correlation with the popularity of the mobile app. Second, we consider the factors related to the information presented in the content (e.g., installation instruction, license, support link) to help visitors to understand the repository and get started on it. We collect the number of links (*R_NumLink*), badges (*R_NumBadge*), words presented in the headings of a readme file (*R_heading*), and license (*P_License*). We expect that a readme file with sufficient information could help visitors get started on the repository and improve its popularity. For instance, license information clearly indicates to visitors how to use a repository (Fan *et al.*, 2021; Laurent, 2004). Links are usually used to point to references for external supporting materials. To collect those factors, for each readme file, we collected the heading information, code blocks, indents, images, lists, and badges. We used the regular expression shown in Table 3 with the Python library *re*. For heading information, we extracted each word in the heading, and applied lemmatization and stemming using Python libraries *spacy* and *nltk* for preprocessing. For other factors, we collected the number of occurrences in each readme file. Note that since a badge is also an image, we further check if a link contains "https://img.shields.io" to determine if the link represents a badge.

For the update dimension, we consider the history of a readme file and examine how it was updated. In this study, we consider the number of updates that a readme file receives (*R_NumUpdate*) and update frequency (*R_UpdateInterval*). Aggarwal *et al.* (2014) found that frequent updates on the documentation of a project have a correlation with its popularity. We expect that frequent updates help maintain a repository stay up to date, therefore improving the popularity of the repository. Note that we count the initialization of a readme file when calculating *R_UpdateInterval* in order to avoid zero in the denominator. To collect the update information, we use Github API (GitHub, 2022b) to collect all commits that involve changes on readme files for each studied repository and extracted the message and the time stamps for each commit. We then calculate each factor accordingly.

There are some other confounders that potentially correlate with the popularity of GitHub repositories. Therefore, we also consider factors related to the repository's meta information, i.e., the age (*P_Age*), the major programming language (*P_ProgramLanguage*), the size (*P_size*), license (*P_License*), the popularity (P_Max/Min/Avg/Median_Tag) as controlling factors when building the model in RQ2, since they are related to the project itself and cannot be modified when editing readme file by the repository owner. Note

Table 3: Regular expressions that are used to extract different pieces of information from readme files.

factor	Regular expression
R_heading	#+ .+\\s
R_NumCodeBlock	`{3}([\S\s]*?)`{3}.*
R_NumIndent	^((?:(?:[ ]{4}|\t).*(\R|$))+)
R_NumLink	\[[^\]\r\n]+\]\([^\)\r\n]+\)
R_NumImage	!\[[^\]\r\n]+\]\([^\)\r\n]+\)
R_NumList	(^{,3}(-|\+|*|[0-9\.])+ [\S\s]*?\r\n\r\n)
R_NumBadge	!\[[^\]\r\n]+\]\([^\)\r\n]+\)

that for popularity, we actually have four factors. We calculate the factors based on the meta information we collected for each repository. We present the studied factors in Table 2.

4. Results of the Research Questions

4.1. RQ1: Is there a relationship between the studied readme file related factors and the popularity of GitHub repositories?

Motivation The readme file of a GitHub repository provides the first expression of the repository to visitors and plays an important role for visitors to get to know the repository. A readme file may have a correlation with the popularity of the GitHub repository. In this RQ, we compare the readme file related features between popular and non-popular GitHub repositories. By understanding this, GitHub project owners may leverage our findings to further improve their projects.

Approach For each of the readme file related numeric features (see details in Section 3.3), we apply the Wilcoxon rank-sum test (Mann and Whitney, 1947) to examine whether the differences in these features between popular and non-popular GitHub repositories are statistically different or not. We also compute Cliff's *delta* to examine the effect size of the difference between the two groups (Cliff, 1993). Note that we interpret $|d|$ as follows: a $|d|$ less than 0.147, between 0.147 and 0.33, between 0.33 and 0.474, and larger than 0.474 is considered as a negligible (N), small (S), medium (M), and large (L) effect size following prior study (Romano *et al.*, 2006). The reason we select the Wilcoxon rank-sum test and cliff's *delta* since they are non-parametric and do not require any assumption on the distribution of two samples.

To understand if there exist differences in the content between the two groups, we compute frequent words that appear in the headings of readme files. We choose to analyze words appearing in headings, since the headings usually summarize the major topics presented in a readme file. We also applied stemming to reduce the size of vocabulary and convert similar words into their original form (e.g., we convert "install", "installed", "installation" into "install"). After the pre-processing, we compare the top 15 frequent words mined from the headings of readme files of the two groups.

Results: All studied numeric readme file-related features have statistically significant differences between popular and non-popular repositories with non-negligible effect size. Table 4 shows the comparison of the studied numeric features between popular and non-popular GitHub repositories, including the mean, median, minimum, and maximum values of each factor. We observe that the two groups of repositories have statistically significant differences (p-value < 0.05) with a non-negligible effect size for all the studied features, in which *R_NumLink*, *R_NumUpdate*, *R_NumUpdate*, *R_Length*, and *R_SinceLastUpdate* have large effect size. Therefore, in general, popular and non-popular GitHub repositories have significant differences in terms of the presented structure, length, and update activities of readme files.

For instance, popular Github repositories (12,396.1 characters) have a four times longer readme file than non-popular repositories (3,115.8 characters) on average. A longer readme file usually suggests more detailed information about the repositories. Meanwhile, popular repositories receive more updates (mean value is 17.8) compared with non-popular repositories (mean value is 6.7). Popular repositories have a mean value of 6.2 code blocks, 10 indents, and 3.4 lists, which are significantly more than non-popular repositories with 2.3 blocks, 5.2 incidents, and 0.02 lists on average. That is said, popular repositories are usually better presented than non-popular ones. For instance, the repository *AXIOS*[3] is a promise-based HTTP client for the browser and node.js. It has 91k stars (as of the date of data collected), which is a very popular GitHub repository. It contains a table of contents to provide the index of the readme file. The readme file describes the features, detailed installing instructions, usage examples, lists of resources, which is very long.

[3] `https://github.com/axios/axios`

Popular and non-popular repositories share similar headings when examining the top 15 most frequent words appearing in headings of readme files. Table 5 presents the top 15 frequent words occurring in the headings of readme files. We notice that 11 out of the top 15 words are overlapped between these two groups. Such observation indicates that the readme files of popular and non-popular repositories share similar sections. For instance, sections related to license (i.e., "license"), how to use/start the repository(i.e., "install", "start", "build", "example", "usage"), contribution ("contribute"), support ("support") commonly shared in the readme files of both groups. More importantly, 4 out of the top 5 words between the two groups are overlapped, i.e., "license", "install", "contribute", and "start". For instance, tensorflow[4] hosts the repository for tensorflow the readme file contains information about installation, license, contribution guidelines, patching guidelines, Continuous build status, resources, and courses. In other words, both popular and non-popular introduce how to use the repository, license, and contributor with top priority. This finding echoes the previous study by Prana *et al.* (2019), in which they observed that 88.5% readme files have a section to introduce how to use the repository, and 52.9% of the readme files have information related to contributor and license.

Table of contents appears more frequently in readme file of popular repositories than non-popular repositories. We also notice that the word "content" appears frequently in popular repositories, while not in non-popular ones. We manually examined several readme files containing "content" and find such readme files usually have a heading "table of contents" to indicate the content included in the readme files. We find that it appears in popular repositories (71 out of 756 popular repositories), while only 11 out of 810 in non-popular repositories.

All studied readme file related features except heading content are statistically significantly different between popular and non-popular repositories with non-negligible effect size. Popular and non-popular repositories share similar headings. While the table of contents appears more frequently in the readme file of popular repositories than non-popular repositories.

[4] `https://github.com/tensorflow/tensorflow`

Table 4: P-value and cliff's $|D|$ for the readme file related features.

	Mean/Median/Min/Max		
Feature	**Popular**	**Non-popular**	**P-value & Cliff's $\|d\|$**
R_NumCodeBlock	6.2/2/0/349	2.3/0/0/73	<0.05 & 0.26 (S)
R_NumIndent	10.0/2/0/366	5.2/0/0//283	<0.05 & 0.15 (S)
R_NumImage	4.3/2/0/171	0.8/0/0/14	<0.05 & 0.41 (M)
R_NumLink	43.5/16/0/1153	5.3/1/0/266	<0.05 & 0.67(L)
R_NumList	3.4/0/0/171	0.02/0.0/0/5	<0.05 & 0.39 (M)
R_NumBadge	1.1/0/0/53	0.14/0/0/8	<0.05 & 0.26 (S)
R_Length	12,396.1/6,459/14/178,059	3,115.8/1,403/11/60,947	<0.05 & 0.63 (L)
R_UpdateInterval	254.7/128/34/4,599	665.9/336/6/4,535	<0.05 & 0.9 (L)
R_SinceLastUpdate	693.4/433/82/3,958	2,401.6/2,471/171/3,714	<0.05 & 0.91(L)
R_NumUpdate	17.8/20/1/30	6.7/3/1/28	<0.05 & 0.686 (L)

Table 5: Top frequently appearing words in the headings of readme files in popular and non-popular repositories. The top 5 words for popular and non-popular repositories are highlighted.

	Popular		**Non-popular**	
Frequent word	**Frequency**	**Rank**	**Frequency**	**Rank**
license	0.313	**1**	0.15	**2**
install	0.29	**2**	0.19	**1**
contribute	0.274	**3**	0.11	**4**
start	0.181	**4**	0.095	**5**
document	0.165	**5**	0.055	15
content	0.161	6		
build	0.156	7	0.087	6
support	0.149	8	0.055	14
example	0.143	9	0.065	9
code	0.136	10		
usage	0.131	11	0.14	**3**
feature	0.131	12	0.063	10
project	0.128	13		
get	0.118	14	0.056	13
learn	0.115	15		
run	0.105		0.072	7
develop			0.0697	8

4.2. RQ2: What are the most important readme file related factors that differentiate popular and unpopular GitHub repositories?

Motivation Among the studied factors, some factors may be more important than others. In this research question, we investigate the most important factors in differentiating two groups of repositories. Our findings probably can provide insights to repository owners, helping them improve their readme files.

Approach To compute the importance of the studied factors, we build a random forest classification model (Cutler *et al.*, 2012) to differentiate popular and non-popular GitHub repositories. The factors that contribute more explanation power are considered to be more important. Below, we elaborate on each step of the model construction, model validation, and variable importance analysis.

Correlation & redundant factor analysis. Before constructing the model, we perform correlation analysis to remove correlated factors. Highly correlated factors can cause multicollinearity problems in our model (Farrar and Glauber, 1967), and turned out to bias the model interpretation. Thus, we perform a correlation analysis to remove highly correlated factors using a variable clustering analysis technique by following prior studies (Rajbahadur *et al.*, 2019; Wang *et al.*, 2018; Fan *et al.*, 2021). First, we use the Spearman rank correlation test to measure the correlation between factors and remove highly-correlated factors (using a cut-off value of 0.7). For each group of the highly-correlated factors, we keep one factor as the representative for the group and then performed a redundancy analysis to remove redundant factors following prior studies (Rajbahadur *et al.*, 2019; Wang *et al.*, 2018; Fan *et al.*, 2021). We ended up with 14 factors (see remaining factors in Table 6).

Model construction. Most of the factors which are numeric and categorical could be directly derived from the repository and used as a feature when building the model. However, one factor *R_heading* is text-based. Because our objective is for explanation and not for prediction, we use permutation variable importance (see below for details) to examine which factors are important. If we directly treated each occurred word in the headings as a feature in the model, it would be difficult to determine the importance of *R_heading* as a whole, since permutation variable importance can only provide the importance for each individual feature (i.e., each word in the heading) rather than the importance of a collection of the words in headings. Therefore, we require a special process on *R_heading*. Following previous studies (Shihab *et al.*, 2013; Mizuno and Hata, 2010), we use a classifier to convert the

heading text into a single numeric factor. More specially, we build a classifier to discriminate between popular and non-popular repositories using the heading text of repositories in the training set. The headings in repositories are divided into tokens using write space, where each token represents a single word. We did not stem the words or remove stop words since words appearing in the headings are usually important. Once the classifier (i.e., *Classifier 1*) is built, we get a probability score for each repository based on its heading text. Then, we build another classifier (i.e., classifier 2) using all factors in Table 2 except *R_heading* together with the probability generated by classifier 1 to discriminate between popular and non-popular repositories.

In this study, we select random forest as our classifiers (for both classifiers 1 and 2). Since our goal here is to conduct model explanation (i.e., understanding which factors are important) rather than prediction. Therefore, we exclude deep learning models since they are hard to explain (Castelvecchi, 2016; Shwartz-Ziv and Tishby, 2017). In general, a model with a higher accuracy indicates more reliable explanation results (Jia *et al.*, 2021). We select random forest since random forest is generally highly accurate (Ghotra *et al.*, 2015; Rajbahadur *et al.*, 2017) and is also robust to outliers since it summarizes the classification results of multiple trees that are learned differently (Cutler *et al.*, 2012). We actually compared the performance of different commonly used classification models from various families on our task, i.e., logistic regression (AUC is 0.90), SVM (AUC is 0.92), decision tree (AUC is 0.89), AdaBoost(0.97), k-nearest neighbors (0.91), random forest (AUC is 0.98). We found that random forest achieved the best performance, therefore we selected random forest for our study. We use the Python package *sklearn.ensemble.RandomForestClassifier* as the implementation of our random forest model. Note that in this study, we consider the factors related to the repository (i.e., *P_Age*, *P_size*, *P_ProgramLanguage*, *P_License*, and *P_Max/Min/Avg/Median_Tag*) as the controlling factors.

Model validation. We use the area under the Receiver Operating Characteristic Curve (i.e., ROC) to assess the explanatory power of the built random forest (Rajbahadur *et al.*, 2017). A random classifier has an AUC (i.e., the area under the curve) of 0.5, while the AUC for a perfect classifier is equal to 1. In practice, most of the classification models have an AUC between 0.5 and 1. In general, an AUC of 0.5 suggests no discrimination (i.e., unable to distinguish between popular and non-popular repositories), 0.7 to 0.8 is considered acceptable, 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding (Mandrekar, 2010). To ensure that our model is not

overfitted and the results are reliable, we perform 10-fold cross validation to validate the performance of the built model.

Variable importance analysis. In this study, we use permutation feature importance (Permutation) to measure the importance of each feature. Permutation is one of the oldest and most popularly used approaches in both machine learning (typically for random forest) and software engineering communities (Rajbahadur *et al.*, 2021, 2017; Janitza *et al.*, 2013; Jiarpakdee *et al.*, 2020). It is defined to be the decrease in a model performance score when a single feature value is randomly shuffled We use the Python implementation *sklearn.inspection.permutation_importance*. We use partial dependence plot (PDP) to show the marginal effect a feature has on the predicted outcome of our built model (Friedman, 2001).

Results: Our model explains our dataset well and has reliable performance. We find that the average AUC of our built classifier is 0.98. An AUC large than 0.7 is generally considered acceptable (Hosmer Jr *et al.*, 2013; Zhou *et al.*, 2020a). The results suggest that our built model explains the relationship between the studied factors and the popularity of a repository well.

After controlling factors related to the repository (e.g., *P_Avg_Tag* and *P_License*), *R_NumList*, *R_NumUpdate*, and *R_UpdateInterval* are statistically significantly important factors that discriminate popular and non-popular repositories. From Table 6, we observe that several repository related factors (i.e., controlling factors) are statistically significantly important to the built model (p-value). For instance, *P_Avg_Tag* is the most important factor to discriminate between two groups of repositories. *P_License* and *P_ProgramLanguage* rank as the second and fourth most important. Besides the controlling factors, we observe that *R_NumList*, *R_UpdateInterval*, *R_NumLink*, *R_NumUpdate*, *R_Length* are statistically significantly to the model. In other words, the presentation and frequent updates are correlated to the popularity of a repository. For instance, CodeXGLUE[5] is a benchmark published by Microsoft for various software engineering tasks (e.g., code summarization and code search). The readme file contains various links to external supporting resources, such as large pre-trained models, other datasets, and related papers, which are helpful for developers to understand and reuse the benchmark, and develop new

[5] `https://github.com/microsoft/CodeXGLUE`

Factor	Variable importance (median)	p-value
P_Avg_Tag	0.49	2.5e-34
P_License	0.22	2.5e-34
R_NumList	0.16	2.5e-34
P_ProgramLanguage	0.11	1.55e-19
R_NumUpdate	0.089	1.9e-06
R_UpdateInterval	0.086	2.7e-16
R_Length	0.047	0.0049
R_NumLink	0.038	0.0003
R_heading	0.022	0.54
R_NumIndent	0.01	0.90
R_NumCodeBlock	0.01	0.62
P_size	0.011	7.6e-08
R_NumImage	0.01	0.22
R_NumBadge	0.007	0.007

Table 6: Variable importance for studies factors that are ordered from the most important to least important.

techniques based on the benchmark.

There is a positive correlation between the frequency and number of readme updates and the likelihood of a repository being popular. As shown in Figure 4, the likelihood of a repository being classified as popular is positively correlated with *R_NumUpdate* and negatively correlative with *R_UpdateInterval*. For *R_NumUpdate*, the likelihood increases sharply from 0 to 7 and gets relatively stable afterward. In terms of *R_UpdateInterval*, the likelihood decreases sharply from 0 to ∼600 and gets stable afterward. Note that smaller *R_UpdateInterval* indicates more frequent updates. Such observations indicate that there is a positive correlation between the frequency and number of readme updates and the likelihood of a repository being popular.

There is a positive correlation between the number of lists and links and the popularity of a repository. Figure 4 shows a similar trend for *R_NumLink* and *R_NumList*, sharply increasing from 0 to a certain value (i.e., 3 for *R_NumList* and ∼30 for *R_NumLink*). Such observations suggest that proper markdown format and providing a proper amount of external support materials probably have a positive correlation with the popularity of repositories.

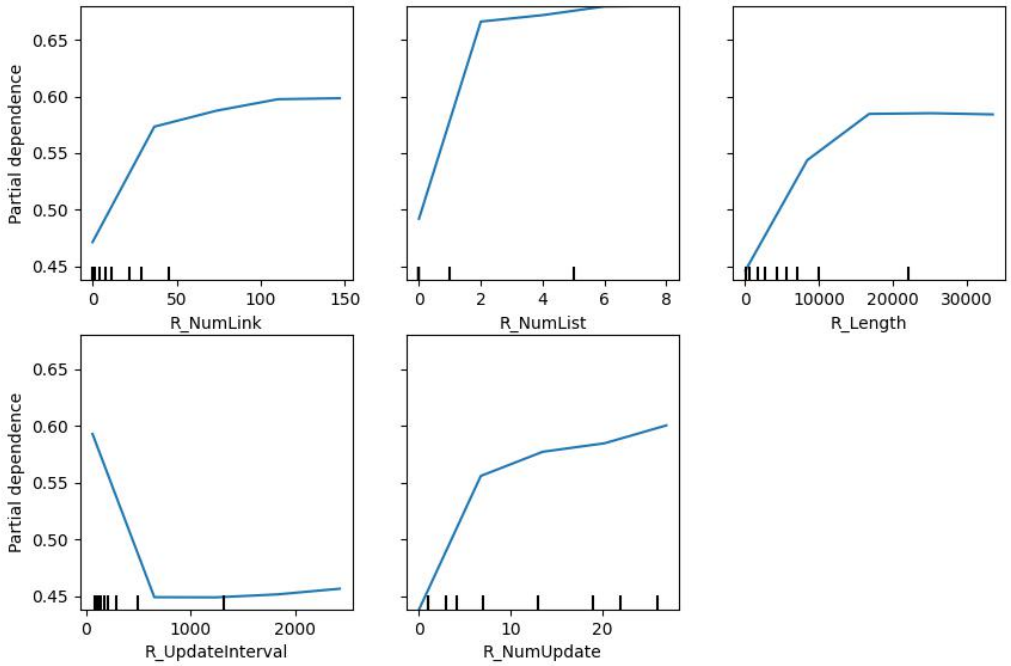

Figure 4: The estimated probability when the values of *R_NumLink*, *R_NumList*, *R_UpdateInterval*, *R_Length*, and *R_NumUpdate* change. Y axis is the probability of a repository being classified as a popular one.

In summary, We find that readme related factors have a high correlation with the popularity of a repository. Therefore, we suggest repository owners should pay more attention to factors related to readme files, such as frequently updating the readme up to date and using proper markdown format to better organize the content in readme files.

After controlling repository-specific factors (e.g., P_Avg_Topic), R_NumList, R_NumLink, R_Length, R_NumUpdate and R_UpdateInterval are statistically significantly important factors that discriminate popular and non-popular repositories. There is a positive correlation between the frequency, number of readme updates, the number of lists and links, and the likelihood of a repository being popular.

4.3. RQ3: What content is updated in readme files between popular and non-popular GitHub repositories?

Motivation As observed in RQ2, factors related to readme update dimension are significantly important for the built model. In particular, *R_UpdateInterval* is the most important factor after controlling repository-specific factors. Therefore, in this section, we explore the update activities in readme files to understand what and when were updated in readme files and compare these activities between popular and non-popular repositories.

Approach To further understand what content was updated in readme files of popular and non-popular repositories, we need to manually examine the updates in the commits. Note that since the number of stars of a repository changes over time, to understand the updates in readme files for popular repositories, we only extract the commits after the repositories become popular (i.e., after receiving at least 100 stars in our case). To do so, we first used star history data provided by Bytebase[6] to obtain the star history of each popular repository, and then identify the moment when the repository achieved at least 100 stars. We then extracted all commits of popular repositories that involved readme file updates after they became popular. Since there are too many commits and it is infeasible for us to manually examine all of them. Therefore, we randomly sampled 100 commits for popular and non-popular repositories, respectively. We use the coding scheme created by

[6]https://star-history.com/

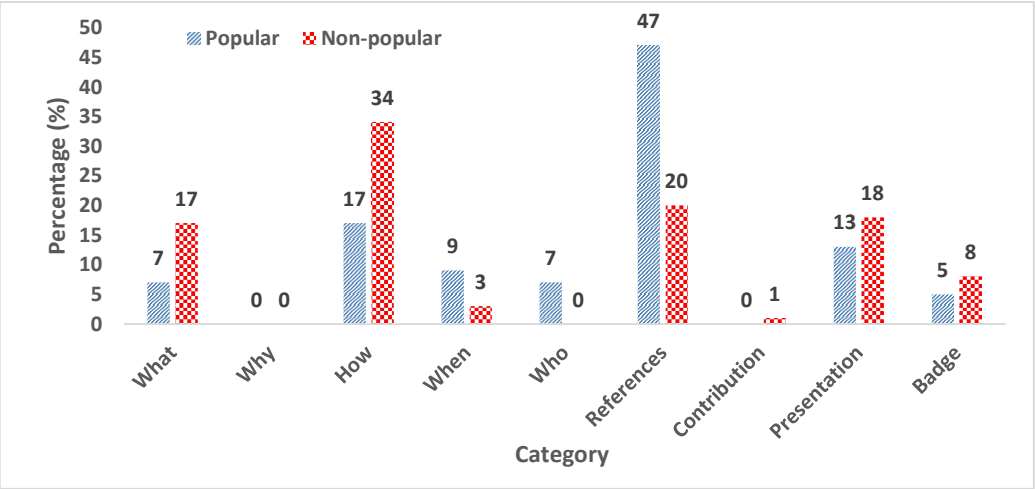

Figure 5: Comparison of updated content in the readme files between popular and non-popular repositories.

prior study (Prana *et al.*, 2019) to label the content updated in readme files. To determine if the existing codes are sufficient for our study, the first author examined the 50 sampled commits with 25 from each group using open code, respectively. After this pilot examination, additional categories are derived. To alleviate the bias from human labeling, the first two authors then label the 50 sampled commits collaboratively. During this phase, the categories are revised and re-fined. Finally, we derived two additional categories "Badge" and "Presentation". Then the first two authors independently applied the categories to the rest of sampled commits and any disagreements until a consensus is reached. During this phase, no new categories were introduced. The inter-rater agreement of this coding process has a Cohen's kappa of 0.81. Table 7 presents the final categories we used to label the updated content in the readme files and example headings.

Results: A larger portion of popular repositories were updated than non-popular repositories. Although we observe that update activities are an important factor for discriminating popular and non-popular repositories, we observe that 34.6% (1728 out of 5,000) of the studied repositories have never been updated since creation. The situation is better in popular repositories, in which only 9% of them have never been updated, compared with non-popular repositories (35%). In terms of *R_UpdateInterval* and *R_NumUpdate* as shown in RQ1, popular repositories' readme

Table 7: Categories of updated content in the readme file.

Category	**Definition**	**Example headings in readme files**
What	Update the introduction of the repository.	Introduction, project background
Why	Update the description of advantages of the repository.	Advantages of the repository, comparison with related work
How	Update the information about how to use the project.	Getting started/quick start, how to run, installation, how to update, configuration, setup, requirements, dependencies, languages, platforms, demo, downloads, errors and bugs
When	Update the description of the status and plan of the repository.	Project status, versions, project plans, roadmap
Who	Update the information about who the project gives credit to.	Project team, community, mailing list, contact, acknowledgment, licence, code of conduct
References	Update the references to external information related to the repository.	API documentation, getting support, feedback, more information, translations, related projects, donation
Contribution	Update the information about how to contribute to the repository.	Contribution guidelines
Badge	Update the badge information for the repository.	NA
Presentation	Fix grammar issues and change markdown format.	NA

files are updated more frequently (254.7 days per update) than non-popular repositories (665.9 days per update).

Most of the updates (47%) were performed to update references in popular repositories, while most updates were performed to update the content of "how" in non-popular repositories (34%). Figure 5 compare the categories of content updated in readme files between popular and non-popular repositories. On one hand, we observe that categories *How*, *References*, and *Presentation* are the top three most frequent categories in both popular and non-popular repositories. Those three categories account for 77% and 72% of popular and non-popular repositories, respectively. More importantly, we observe both in popular and non-popular projects, developers make significant efforts to update references, such as API documentation and related projects. For example, one commit was made to update a broken URL of API documentation to resolve an issue raised by developers[7]. This finding is compatible with prior studies on software knowledge platforms in which the authors found that a remarkable proportion of links are obsolete (Zhang *et al.*, 2019; Liu *et al.*, 2020). For instance, Zhang *et al.* (2019) found that 11.9% of the 5.5 million links in Stack Overflow answers are broken. Developers in both groups of repositories edit frequently the information on how to use the repositories, which is expected. One possible explanation is that almost all the repositories have such information and they are easy to become outdated (Prana *et al.*, 2019).

A remarkable portion of updates on readme files were performed to improve the presentation and update badges for both popular and non-popular repositories. Developers also make efforts to improve the presentation of readme files (13% and 18% for popular and non-popular repositories). For instance, in a commit[8], the contributor changed the format of section titles to make them more consistent. Based on our results in Section 4.2, the factors related to the presentation of readme files are important factors for discriminating between popular and non-popular repositories. In other words, the presentation of readme files is important for the popularity of a repository. 5% and 8% of updates on readme files are related to badge (i.e., category *Badge*) for popular and non-popular repositories, respectively.

[7] `https://github.com/cristaloleg/go-advice/commit/c78d6c06d5327d731f61bbe416eeec2c28e79586`

[8] `https://github.com/emberjs/ember.js/commit/fb7022b8f4af64534470b002fe91a4da9b9d96b5`

Various badges could be added to the readme file of a repository to indicate the standard/property of a repository, e.g., a 100% coverage badge could be added to indicate that 100% of the code in the repository is covered by its test cases. For instance, in a commit of openframeworks [9], the contributor updated the badge status (e.g., changed the status of build on Linux 64 & Arm from "errored" to "passing"). They also changed the name of the badge for each platform to make the badge more specific (e.g., from "build" to "build-linux64-and-arm"). We compare the ratio of repositories in two groups that have at least one badge. Interestingly, we find more popular repositories have at least one badge (30%) than non-popular repositories (6.7%).

Most of the updates (47%) were performed to update references in popular repositories, while most updates were performed to update the content of "how" in non-popular ones (34%). A remarkable portion of updates on readme files were performed to improve the presentation and update badges for both popular and non-popular repositories.

5. Discussion

In this section, we discuss the implications of our findings and threats to validity.

5.1. Implications of Our Findings

We suggest the GitHub repository contributors use proper markdown to organize the content of the readme file properly. In RQ2, we observe that *R_NumLink* and *R_NumList* are significantly important factors and have a positive correlation with the popularity of a repository. We also observe that the table of contents are appeared more frequently in the readme file of popular repositories than in non-popular repositories, especially when the readme file is long. Therefore, we suggest contributors should pay attention to using proper markdown to organize the content of readme files.

We suggest the GitHub repository contributors make continuous efforts to keep readme files up to date. In RQ2, we observe that

[9] `https://github.com/openframeworks/openFrameworks/commit/faba5895d2687588ee0512ed90fb5288c8f3de1d`

R_NumUpdate and *R_UpdateInterval* are statistically significantly important factors in the built model. Although our study does not reveal causation, at least we observe the frequency/number of readme updates positively correlates with the likelihood of a repository being popular. In other words, updates on the readme file are important. In RQ3, we observe that a remarkable proportion of updates were performed to update references and content related to "how". Prior studies already prove that a good quality readme file can reduce the learning curve of users to get started and contribute to a repository (Steinmacher *et al.*, 2014). Therefore, contributors should pay attention to maintaining the repository and make all content up to date.

In summary, the quality of the readme file is positively correlated with the popularity of repositories. Developers should pay attention to improving their readme files.

5.2. Future research direction

Tools are encouraged to be developed to support the automatic update for readme files. In RQ3, we observe that most of the updates (47%) in popular repositories were performed to update references due to various reasons, such as broken links and obsolescence. We also observe that developers frequently update the information on how to use the repository due to their obsolescence. Developers make significant efforts to maintain the references in their readme files. Therefore, future research is encouraged to develop tools to support such updates. One solution is to develop approaches to detect such obsolete references and content in the readme file during pushing a commit and recommend proper update solutions similar to prior research in just-in-time commit message generation (Xu *et al.*, 2019; Liu *et al.*, 2018). We also observe developers update the readme file to improve presentation, e.g., improving the formatting. Recently, large language models (e.g., GPT-4) have been proven in various NLP tasks (Bubeck *et al.*, 2023; OpenAI, 2023). Future research is encouraged to leverage large language models for improving the presentation of readme files.

Future research is encouraged to generate a readme template for a repository automatically. In RQ1, we observe that popular and non-popular repositories share similar headings (e.g., usage and installation). Certain topics are frequently contained both in the readme file of popular and non-popular repositories. In other words, the readme files of GitHub repositories usually follow patterns containing certain pieces of information.

Therefore, there is potential to generate a readme template automatically when a repository description is given using historical readme files as the training data using machine learning techniques, such as generative models (Radford *et al.*, 2018; Nie *et al.*, 2018). We encourage future research to investigate this.

5.3. Threats to Validity

Internal Validity One threat to internal validation relates to the categorization of the studied repositories as popular and non-popular ones, in which we use a hard-cut 0 and 100 as the threshold. However, different thresholds may lead to different observations. To alleviate this threat, we build a random forest model using different thresholds. Instead of using the number of stars directly, we use percentages following prior studies (Zhou *et al.*, 2020b; Wang *et al.*, 2018). More specifically, we sort the repositories based on their starts and consider the top k% and bottom k% of the repositories as popular and non-popular. We tested three values 10, 20, and 30 for k. The findings still hold. The findings still hold. The *R_NumList*, *R_NumLink*, and *R_UpdateInterval* are still the most important features after controlling repository-related features.

One internal threat is that in RQ1, we use the most frequent words appearing in the headings as a proxy to estimate the content of readme files. Some readme files probably do not have headings. However, we find the proportion of such cases is low, i.e., 6% in popular but 29% in non-popular. Another internal threat is that in RQ1, we look at the top 15 most frequent words appearing in the headings to compare the content of readme files between popular and non-popular repositories. The findings may vary when we select different thresholds. To alleviate the threat, we select different thresholds (i.e., top 20 and 30) and we find the overlap between the two groups is 70% and 50%, respectively.

Another threat is that in data preparation, we did not consider outliers and their impact, typically the impact of the extremely popular repositories. To alleviate this threat, We used the three-sigma rule of thumb to identify outliers (Pukelsheim, 1994). For popular repositories, we identify 17 repositories that have more than 86,054 stars as outliers. For non-popular repositories, since all the repositories have zero star and we do not identify any outliers. After removing those outliers, we re-ran the analysis in RQ1 and RQ2 and our findings still hold.

In RQ2, we study the association between the studied features and the likelihood of a repository being popular. However, we can not conclude the causation between them. Future research is encouraged to study their causation of them.

In RQ3, we involved human labeling. To reduce the bias from human labeling, each commit was labeled by two of the authors, and discrepancies were discussed until a consensus was reached. We also showed that the level of inter-rater agreement of the qualitative study is high (i.e., the value of Cohen's kappa is 0.81).

External Validity One external threat is that it is not clear whether our findings still hold on to other repositories (e.g., closed source repositories and open source repositories on other platforms). Although we performed the study on the most popular repository hosting platform (i.e., GitHub), future research is encouraged to replicate our analysis on other repositories.

Another threat is regarding the factors that there might be additional factors that could be associated with the popularity of a repository. In this study, we consider 6 factors related to the content of readme files, 3 factors related to the update activities on readme files, and 8 factors related to the repository as our controlling factors. There might be other factors associated with the popularity of a repository. However, our results show that the explanatory power of our models is high (0.98) when using the studied factors. Future studies should consider more features.

6. Conclusion and future work

In this study, we analyze the readme files of 5000 GitHub repositories across different domains and languages. We investigate the relationship between readme file related factors and the popularity of GitHub repositories. We observe that all studied readme file related features (e.g., the number of links and images, updates frequency) except heading content are significantly different between popular and non-popular repositories with non-negligible effect size. Popular and non-popular repositories share similar headings. We also observe the number of lists and the frequency of updates are significantly important factors that discriminate between popular and non-popular repositories and they positively correlate with the likelihood of a repository being popular. In summary, the quality of the readme file (e.g., proper organization of content and ensuring content is up to date) is positively correlated with the popularity of repositories. The most of the updates were performed to update

reference in popular repositories, while the most updates were performed to update the content of "how" in non-popular ones. A remarkable portion of updates on readme files were performed to improve the presentation and update badges for both popular and non-popular repositories. Developers should pay attention to improving the readme file, such as improving the organization of content and ensuring the readme is up to date.

As the one of the first studies on this direction, we provide certain actionable insights for contributors on GitHub and future research. We encourage future research to verify those insights.

References

Aggarwal, K., Hindle, A., and Stroulia, E. (2014). Co-evolution of project documentation and popularity within github. In *Proceedings of the 11th working conference on mining software repositories*, pages 360–363.

Bao, L., Xia, X., Lo, D., and Murphy, G. C. (2019). A large scale study of long-time contributor prediction for github projects. *IEEE Transactions on Software Engineering*, **47**(6), 1277–1298.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of github repositories. In *2016 IEEE international conference on software maintenance and evolution (ICSME)*, pages 334–344. IEEE.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., *et al.* (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Castelvecchi, D. (2016). Can we open the black box of ai? *Nature News*, **538**(7623), 20.

Cliff, N. (1993). Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, **114**(3), 494.

Coelho, J., Valente, M. T., Silva, L. L., and Shihab, E. (2018). Identifying unmaintained projects in github. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10.

Cutler, A., Cutler, D. R., and Stevens, J. R. (2012). Random forests. In *Ensemble machine learning*, pages 157–175. Springer.

Fan, Y., Xia, X., Lo, D., Hassan, A. E., and Li, S. (2021). What makes a popular academic ai repository? *Empirical Software Engineering*, **26**(1), 1–35.

Farrar, D. E. and Glauber, R. R. (1967). Multicollinearity in regression analysis: the problem revisited. *The Review of Economic and Statistics*, pages 92–107.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Gharehyazie, M., Ray, B., and Filkov, V. (2017). Some from here, some from there: Cross-project code reuse in github. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 291–301. IEEE.

Gharehyazie, M., Ray, B., Keshani, M., Zavosht, M. S., Heydarnoori, A., and Filkov, V. (2019). Cross-project code clones in github. *Empirical Software Engineering*, **24**(3), 1538–1573.

Ghotra, B., McIntosh, S., and Hassan, A. E. (2015). Revisiting the impact of classification techniques on the performance of defect prediction models. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 789–800. IEEE.

GitHub (2022a). About READMEs. `https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes`.

GitHub (2022b). REST API. `https://docs.github.com/en/rest`.

GitHub (2022c). Where the world builds software. `https://github.com/about`.

Google (2020). Use BigQuery to query GitHub data. `https://codelabs.developers.google.com/codelabs/bigquery-github`.

Han, J., Deng, S., Xia, X., Wang, D., and Yin, J. (2019). Characterization and prediction of popular projects on github. In *2019 IEEE 43rd annual computer software and applications conference (COMPSAC)*, volume 1, pages 21–26. IEEE.

Hassan, F. and Wang, X. (2017). Mining readme files to support automatic building of java projects in software repositories. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 277–279. IEEE.

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied logistic regression*, volume 398. John Wiley & Sons.

Ikeda, S., Ihara, A., Kula, R. G., and Matsumoto, K. (2019). An empirical study of readme contents for javascript packages. *IEICE TRANSACTIONS on Information and Systems*, **102**(2), 280–288.

Janitza, S., Strobl, C., and Boulesteix, A.-L. (2013). An auc-based permutation variable importance measure for random forests. *BMC bioinformatics*, **14**(1), 1–11.

Jia, Y., Frank, E., Pfahringer, B., Bifet, A., and Lim, N. (2021). Studying and exploiting the relationship between model accuracy and explanation quality. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21*, pages 699–714. Springer.

Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P. S., and Zhang, L. (2017). Why and how developers fork what from whom in github. *Empirical Software Engineering*, **22**(1), 547–578.

Jiarpakdee, J., Tantithamthavorn, C., and Treude, C. (2020). The impact of automated feature selection techniques on the interpretation of defect models. *Empirical Software Engineering*, **25**(5), 3590–3638.

Kimble, J. (1992). Plain english: A charter for clear writing. *TM Cooley L. Rev.*, **9**, 1.

Koskela, M., Simola, I., and Stefanidis, K. (2018). Open source software recommendations using github. In *International Conference on Theory and Practice of Digital Libraries*, pages 279–285. Springer.

Laurent, A. M. S. (2004). *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software*. " O'Reilly Media, Inc.".

Liu, J., Xia, X., Lo, D., Zhang, H., Zou, Y., Hassan, A. E., and Li, S. (2020). Broken external links on stack overflow. *arXiv preprint arXiv:2010.04892*.

Liu, Y., Noei, E., and Lyons, K. (2022). How readme files are structured in open source java projects. *Information and Software Technology*, **148**, 106924.

Liu, Z., Xia, X., Hassan, A. E., Lo, D., Xing, Z., and Wang, X. (2018). Neural-machine-translation-based commit message generation: how far are we? In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 373–384.

Mandrekar, J. N. (2010). Receiver operating characteristic curve in diagnostic test assessment. *Journal of Thoracic Oncology*, **5**(9), 1315–1316.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.

McDonald, N. and Goggins, S. (2013). Performance and participation in open source software on github. In *CHI'13 extended abstracts on human factors in computing systems*, pages 139–144.

Mizuno, O. and Hata, H. (2010). An integrated approach to detect fault-prone modules using complexity and text feature metrics. In *Advances in Computer Science and Information Technology*, pages 457–468. Springer.

Munaiah, N., Kroh, S., Cabrey, C., and Nagappan, M. (2017). Curating github for engineered software projects. *Empirical Software Engineering*, **22**(6), 3219–3253.

Nie, W., Narodytska, N., and Patel, A. (2018). Relgan: Relational generative adversarial networks for text generation. In *International conference on learning representations*.

OpenAI (2023). Gpt-4 technical report.

Prana, G. A. A., Treude, C., Thung, F., Atapattu, T., and Lo, D. (2019). Categorizing the content of github readme files. *Empirical Software Engineering*, **24**(3), 1296–1327.

Pukelsheim, F. (1994). The three sigma rule. *The American Statistician*, **48**(2), 88–91.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Rajbahadur, G. K., Wang, S., Kamei, Y., and Hassan, A. E. (2017). The impact of using regression models to build defect classifiers. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 135–145. IEEE.

Rajbahadur, G. K., Wang, S., Kamei, Y., and Hassan, A. E. (2019). Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering. *IEEE Transactions on Software Engineering*, **47**(7), 1414–1430.

Rajbahadur, G. K., Wang, S., Ansaldi, G., Kamei, Y., and Hassan, A. E. (2021). The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Transactions on Software Engineering*.

Romano, J., Kromrey, J. D., Coraggio, J., and Skowronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In *annual meeting of the Florida Association of Institutional Research*, volume 177, page 34.

Sharma, A., Thung, F., Kochhar, P. S., Sulistya, A., and Lo, D. (2017). Cataloging github repositories. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 314–319.

Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E., and Matsumoto, K.-i. (2013). Studying re-opened bugs in open source software. *Empirical Software Engineering*, **18**(5), 1005–1042.

Shwartz-Ziv, R. and Tishby, N. (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

Steinmacher, I., Gerosa, M. A., and Redmiles, D. (2014). Attracting, onboarding, and retaining newcomer developers in open source software projects. In *Workshop on Global Software Development in a CSCW Perspective*.

Tian, Y., Nagappan, M., Lo, D., and Hassan, A. E. (2015). What are the characteristics of high-rated apps? a case study on free android applications. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pages 301–310. IEEE.

Treude, C., Middleton, J., and Atapattu, T. (2020). Beyond accuracy: assessing software documentation quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1509–1512.

Trockman, A., Zhou, S., Kästner, C., and Vasilescu, B. (2018). Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *Proceedings of the 40th International Conference on Software Engineering*, pages 511–522.

Wang, S., Chen, T.-H., and Hassan, A. E. (2018). Understanding the factors for fast answers in technical q&a websites. *Empirical Software Engineering*, **23**(3), 1552–1593.

Weber, S. and Luo, J. (2014). What makes an open source code popular on git hub? In *2014 IEEE International Conference on Data Mining Workshop*, pages 851–855. IEEE.

Xu, S., Yao, Y., Xu, F., Gu, T., Tong, H., and Lu, J. (2019). Commit message generation for source code changes. In *IJCAI*.

Zhang, H., Wang, S., Chen, T.-H., Zou, Y., and Hassan, A. E. (2019). An empirical study of obsolete answers on stack overflow. *IEEE Transactions on Software Engineering*, **47**(4), 850–862.

Zhang, Y., Lo, D., Kochhar, P. S., Xia, X., Li, Q., and Sun, J. (2017). Detecting similar repositories on github. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 13–23. IEEE.

Zhou, J., Wang, S., Bezemer, C.-P., and Hassan, A. E. (2020a). Bounties on technical q&a sites: a case study of stack overflow bounties. *Empirical Software Engineering*, **25**(1), 139–177.

Zhou, J., Wang, S., Bezemer, C.-P., Zou, Y., and Hassan, A. E. (2020b). Studying the association between bountysource bounties and the issue-addressing likelihood of github issue reports. *IEEE Transactions on Software Engineering*.

Zhu, J., Zhou, M., and Mockus, A. (2014). Patterns of folder use and project popularity: A case study of github repositories. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–4.