IEICE TRANS. INF. & SYST., VOL.E102–D, NO.2 FEBRUARY 2019

PAPER

An Empirical Study of README contents for JavaScript Packages

Shohei IKEDA[†], ***Nonmember***, **Akinori IHARA**[†,††a)], ***Member***, **Raula Gaikovina KULA**[†], ***Nonmember***, ***and*** **Kenichi MATSUMOTO**[†], ***Fellow***

SUMMARY Contemporary software projects often utilize a README.md to share crucial information such as installation and usage examples related to their software. Furthermore, these files serve as an important source of updated and useful documentation for developers and prospective users of the software. Nonetheless, both novice and seasoned developers are sometimes unsure of what is required for a good README file. To understand the contents of README, we investigate the contents of 43,900 JavaScript packages. Results show that these packages contain common content themes (i.e., 'usage', 'install' and 'license'). Furthermore, we find that application-specific packages more frequently included content themes such as 'options', while library-based packages more frequently included other specific content themes (i.e., 'install' and 'license').

key words: *documentation, README, association rule mining, JavaScript packages*

1. Introduction

To encourage prospective users and interested developers to write documentation, it is common practice for Open Source Software (OSS) projects to release software artifacts (i.e., source code, configuration files and documentation) through platforms such as GitHub. Some projects release a meta-file document called README, which typically includes a summary of the most useful and updated information, such as an install guide and usage examples. This is especially crucial for tracking changes once newer versions get released. In fact, all GitHub hosted projects present the README on their front page [1].

Developers often struggle to write documentation [2]. A large-scale GitHub survey* conducted in June 2017, reported that although software documentation is highly valued, it is frequently overlooked. Furthermore, most respondents (approximately 93%) complained that most documentation is either incomplete or outdated. In the survey, 60% of contributors said that they rarely or never contribute to documentation. Related studies also confirm that developers struggle to write documentation. Abebe et al. [3] advised developers to note several content themes such as title, system overview, resource requirements, installation, and addressed issues (i.e., new features, bug fixes, and improvements) as caveats in the release note. Moreno et al. [4] reported that developers find it difficult to summarize a release note because it has several content themes, such as fixed bugs, new features, and the improvement of existing features. They proposed an approach to automatically generate release notes. Similarly, other works [5], [6] investigated the relationship between source code (i.e., API, code examples) and documentation. In terms of README files, Hassan et al. [7] proposed an approach to extract a build command, while Zhang et al. [8] used this approach to identify systems with similar functions.

A README file contains key documentation patterns for developers, especially when uncovering documentation patterns specific to the types of software. For instance, library-specific projects (i.e., projects used by other applications as third-party libraries) may write their README file differently in application-specific projects (i.e., projects used by end-users).

In this study, we would like to understand the extent to which developers write and maintain their README files. We conduct an empirical case study that analyzes over 43,900 packages belonging to the npm JavaScript ecosystem in GitHub. In particular, we investigate (i) what constitutes typical content themes and (ii) whether content themes indicate the type of a package (i.e., library-specific vs. application-specific). In this novel study, we learned the following valuable lessons along the way:

- **Lesson 1: It is useful to build and summarize a taxonomy of 20 README content themes, which are used by more than 1% of packages.** - From over 30,000 content variations, we used a semi-automatic method to build a taxonomy of README content themes.
- **Lesson 2: "Usage", "Install", and "License" are common README content themes.** - This result complements known guidelines for writing good documentation. We also found that less apparent README content themes include "API", "Test", and "Todo", are used in 10%–24% of packages.
- **Lesson 3: Our study shows that "Install" and "License" are likely content themes for library-specific packages, while the "Option" content theme is more common for application-specific packages.** - "Install" (i.e., 40% packages) and "License" (i.e.,

Manuscript received February 22, 2018.
Manuscript revised September 9, 2018.
Manuscript publicized October 24, 2018.

[†]The authors are with Graduate School of Information Science, Nara Institute of Science and Technology (NAIST), Ikoma-shi, 630–0192 Japan.
[††]The author is with the Faculty of System Engineering, Wakayama University, Wakayama-shi, 640–8510 Japan.
a) E-mail: ihara@sys.wakayama-u.ac.jp

DOI: 10.1587/transinf.2018EDP7071

*Open Source Survey: http://opensourcesurvey.org/2017/

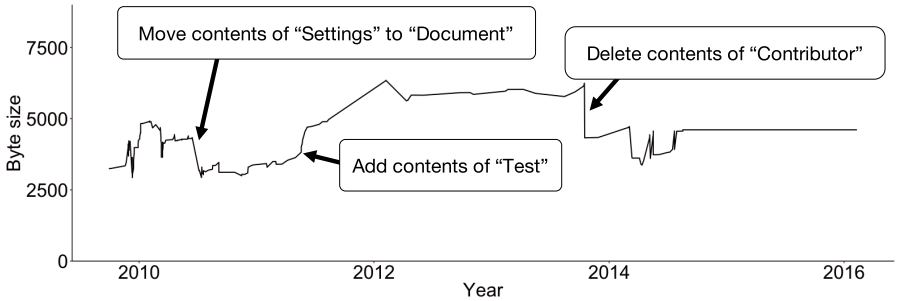

Fig. 1 Illustrative example of evolving `README` file for the `express` package. Note that we measure the file size (in bytes) as a measure of changes.

Table 1 Guideline for writing a `README` file.

Content theme	GitHub	18F	OSCON2015	Summary
Overview	✓	✓	✓	Description of what the project is for and how useful.
Install	✓	✓	✓	Instructions for how to develop, use.
License	✓	✓	✓	List for where your team can ask for contact information.
Contribution	✓	✓	✓	Instructions for how people can help clarify the documentation.
Support	✓	✓	✓	List the contact information for your team where to ask questions.
Author	✓			List for who maintains and contributes to your project.
Usage		✓	✓	List of code sample and config tips.
Release history			✓	List of changes.
Product	✓			Description of code for conduct.

20% packages) are common for npm libraries, while nodejs application packages included the option content themes (i.e., 10% packages).

We conclude that `README` files reveal insights such as project practices and product changes. Such information especially assists especially the novice developer.

This paper is laid out as follows. Section 2 describes the background and motivation of this study. Section 3 provides the dataset to conduct our empirical study. Section 4 presents answers to each of the two research questions proposed in this study. Section 5 discusses our findings. Section 6 presents threats to validity. Finally, Sect. 7 concludes the paper and presents our future work.

2. Motivation & Research Overview

2.1 Illustrative Example & Key Assumptions

Co-founder of GitHub, Tom Preston-Werner recently highlighted the importance of the `README` file, coining Readme Driven Development (RDD)[†] as an important subset of Document Driven Development. In this paper, our motivation is to investigate the following assumptions:

- `README` file is a reliable source of important documentation changes and content themes of the project.
- `README` content themes follow some useful guidelines and may be indicative of its project type.

Figure 1 illustrates an example of how a `README` changes over time and is indicative of other changes. This example shows the JavaScript `express`[††] package. In detail, `express` added content themes of “Test” in 2011. For example, `express` moved the content theme of “Settings” to “Documentation” linking to the official website in 2010. Later, they deleted the content theme of “Contributor”. Interestingly, in a preliminary exploration of 119,093 npm packages, we found on average that a `README` was updated up to 7 times.

Table 1 shows the existing guidelines that hint at the content theme. These guidelines are taken from the following sources:

- *GitHub*[†††] project introduces the content themes which the `README` file typically includes.
- *18F*[††††] project is a digital service agency which introduced “Making `README`s readable”.
- *OSCON2015*[†††††] is an international conference for open source development. Key-note speaker, Mr. Mike Jang explained how open source projects failure to attract users due to poor `README` quality. He later introduced 10 key content themes.

As shown in Table 1, we find that key information such as “Overview”, “Install”, “License”, “Contribution”, “Support”, “Author”, “Usage”, “Release history”, and “Product” are perceived as vital for any software projects.

[†]Readme Driven Development: http://tom.preston-werner.com/2010/08/23/readme-driven-development.html

[††]express: https://www.npmjs.com/package/express

[†††]GitHub Help -About READMEs-: https://help.github.com/articles/about-readmes/

[††††]18F Open Source Style Guide: https://open-source-guide.18f.gov/making-readmes-readable/

[†††††]O’Reilly Open Source Convention: OSCON, July 20–24, 2015 in Portland, OR https://conferences.oreilly.com/oscon/open-source-2015

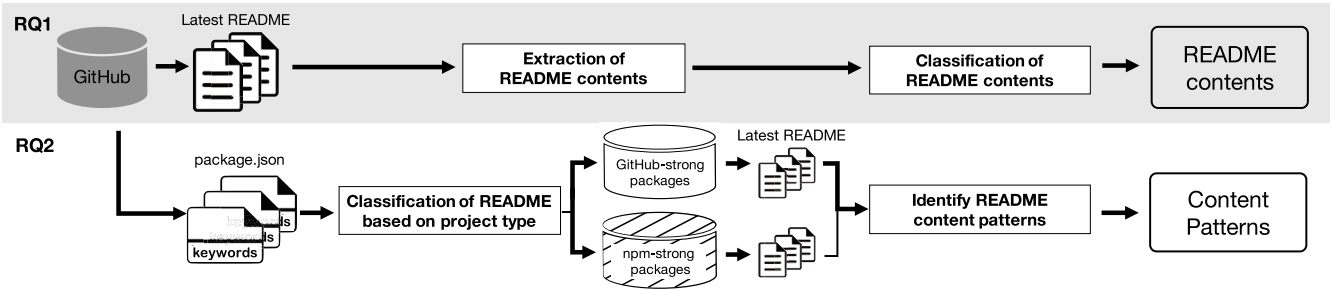

Fig. 2 Procedure of this study.

2.2 Research Questions

Our goal is to understand the content themes. We therefore investigate the extent to which key content themes (i.e., as found in the guidelines) appear in the `README`. As shown in Fig. 2, we formulate the following research questions to guide our study.

- RQ_1**: What do developers write in a `README` file?** As shown in Table 1, we would like to confirm what content themes constitute a `README` file. This may provide information for crucial documentation and will confirm the guidelines for good documentation.
- RQ_2**: Does the type of project affect how developers write their `README` file?** We investigate whether or not projects write `README` files based on their project type. For example, we speculate that some application frameworks (e.g., `express`) or the plugin tools (e.g., `gulp`) projects would prioritize purpose and usage of the software (e.g., sample code, example of option).

To answer our research questions, we perform an empirical study on real-world projects. Specifically, we perform a case study. For RQ_1, we first extract and conduct a content theme analysis of `README` files. Then, for RQ_2, we investigate the relationship between the type of project and the content themes.

3. Data Collection

In this section, we describe both the data extraction and data preprocessing method for the empirical study. The final dataset will be used to evaluate the two research questions.

3.1 Data Extraction

Our study targets a `README` file for JavaScript packages. Specifically, we target on JavaScript projects that belong to the npm ecosystem[†], consisting of packages that run on the nodeJS platform. Packages include libraries (e.g., `react`), frameworks (e.g., `express`), command line tools (e.g., `browserify`), and plug-in-supporting tools to build applications (e.g., `grunt`, `glup`). To support searching these packages, the npm repository adds a keyword tag to each package to explain the features of the package. For example, the `express` package has "framework", "web" and "express".

Table 2 Summary of Extracted Datasets.

Extraction Snapshot of Projects	July 2008–July 2016
Extracted `README` files	153,857 packages
`README` files after preprocessing	43,911 packages
Content themes	30,939 content themes

We use the same process described by Wittern et al. [9] to extract similar datasets for libraries and applications. In detail, we query the npm registry[††] for all npm packages that were hosted and available on GitHub. We then extract the `README` files for all the packages. Finally, we collected 153,857 `README` files as shown in Table 2.

As shown in Fig. 2, our extracted dataset also consists of an extraction of the project type. Hence, for each project, we extract tagged keywords from the package.json metafile. In the Wittern et al. study [9], the authors showed that keywords may be indicative of project type, which is needed to answer RQ_2.

3.2 Data Preprocessing

To ensure the quality of the dataset, we perform filtering to remove the noise in the dataset. After data extraction, we are able to collect 153,857 `README` files that are used "md" and "markdown" as a file extension. Since most `README` files are written in the markdown format and we plan to use English as our main language of analysis, we use the `unicodedata` library to filter out files which did not use the markdown format (i.e., we exclude files written in Japanese, Cyrillic, or Hangul). Then, there are 141,933 `README` files. Furthermore, to ensure that we capture all initial commits, we only include projects that were created within the three year period of our analysis (i.e., 2013~2016) and passed more than one year from the last update of the README. Because we control the threshold to keep latest projects and to filter out the incompleted README files. Hence, we left 43,911 `README` files after preprocessing.

For RQ_2, we further separate our dataset into GitHub-

[†]npm: https://www.npmjs.com/

[††]accessible on July 1–15, 2016 at https://registry.npmjs.org/-/all

strong and npm-strong types of projects as application-specific and library-specific packages. Details of this approach are explained in Sect. 4.2.

4. Empirical Study

In this section, we evaluate the two research questions proposed in Sect. 2.2. For each research question, we describe the approach and their results.

4.1 RQ1: What do developers write in a README file?

(1) Approach

To answer RQ1, we perform an analysis of the README file contents. Our analysis consists of two steps:

- *(Step 1) Extraction of README content themes.* Our key assumption is that headlines in the README file are indicative of important content themes. Since we find that README files often use a header as a summary of their content themes, we perform the following:
  (Step 1a) Extracting Headlines. Targeting the levels 1 and 2 (i.e., $h1$ and $h2$) headlines, we extract 79,898 headlines using this technique. Hence, using the markdown format, we can extract headlines using the syntax (h1: # or ===, h2: ## or ---).
  (Step 1b) Mapping Headline to Content. Since a headline is a natural language, variations, and spelling inconsistencies cause noise in the dataset. For example, developers use "How to Install?", "installing", "Installation" to summaries the "Install" content theme. We use the stemming technique from the language processing package (i.e., nltk package in Python) to normalize and clean noise in the data. The nltk package is well-known and provides a high accuracy of software engineering datasets [10].
  (Step 1c) Merging Similar Content Themes. To further reduce the noise in the content theme dataset, we merge content themes that contain manually merged content themes with similar or related meanings. For example, we conclude that content themes "Getting Started", "To setup", and "Download" should be merged into the "Install" content theme. In this study, the first author, second author, and third author firstly make clusters to merge content themes with each other. Next, if there are the content themes in the different clusters between the authors, we start a discussion to reach consensus on common content themes.
- *(Step 2) Classification of README content themes.* Based on the results of Step 1, we display the frequency count of each content theme and its coverage (i.e., the percentage of systems using each content theme).

Using our approach, we extracted 30,939 content themes from 69,869 headlines from the README files (i.e., Steps 1a and 1b). Table 3 shows an example of Step 2. Furthermore, Table 4 shows some exceptions and adjustments that we encountered to the conventional natural language stemming approach. For instance, using the default settings, the headline "who are we?" would be removed (i.e., Step 1b).

Table 3 Example of our cleaning (i.e., using stemming) of the headlines. In this data processing, we are able to map the variations of headlines to the install content theme.

Headline Variations	Content Theme
How to Install?	Install
installing	
Install it	
Installation	
INSTALL	
1. Installing	

Table 4 Example of stopword exceptions in the mapping headline to content theme (Step 2).

Stopword token	Content Theme	Example of Headline
to do	todo	To do
how	usage	How to
who	author	Who are we?
from		From
more	document	More ....
other		Others
about	overview	ABOUT
what		What the ...?
that		What is that?
can		What can I do?

By merging the more frequent content themes and filtering out the less common content themes (Step 1c and Step 2), we finally ended up with the 20 most frequent content themes used by more than 1% of projects from the 30,939 content themes (i.e., Step 1c). In a semi-automatic approach, we incrementally filter content themes not frequently appearing (i.e., Step 2).

(2) Result

Observation 1 — Many software projects (36.22%–60.83%) contain "Usage", "Install" and "License" content themes in their README files.

Table 5 shows README content themes which we merged with the different headlines. These results also coincided with the recommended guideline content themes (i.e., Table 1), which we show using a check mark (✓).

We found that the top three README content themes are "Usage" (i.e., 60.83%), "Install" (i.e., 59.43%), and "License" (i.e., 36.22%). However, we find that developers often use different variations of the same word to explain each content theme. For instance, developers may use "Example", "Hello World", and "Grunt tasks" content theme keywords as "Usage". Furthermore, while we knew that "License" is a common header, only 36.22% of systems note it in their README files. We suspected that developers may place license information in other meta-files (i.e., package.json) or as a separate document file named "LICENSE".

Observation 2 — Results confirm that README content themes are targeted at its end users.

"Usage" and "Install" represent how to use the sys-

Table 5 A taxonomy of common content themes in README files (note that PR=Packages/All packages). Note that we use the Top 6 content themes for analysis in RQ_3.

Rank	Merged content theme	Packages (PR)	Guideline from Table 1	Description	Content Theme
1	Usage	26,758 (60.83%)	✓	Basic usage example of the project	Usage, Basic Usage, How to Use, Use Case, Methods, Screenshot, Examples, Quick Examples, Tips, Syntax, Sample, Hello World, Grunt tasks
2	Install	26,142 (59.43%)	✓	How to install the project	How to Install?, Installation, Getting Started, Get started now!, To setup, Download, Initialization, Instructions, npm, node.js
3	License	15,932 (36.22%)	✓	Type of license applied to the project	LICENSE, MIT License, Unlicense, License, Copyright, Legal
4	API	10,675 (24.27%)		API list of the project	API, API References, API Documents, Command Line, CLI, Build, Events, Constructor, Action, To run, Objects, Interface, Commands, Function, Execute
5	Option (Product)	10,459 (23.78%)	✓	Option list of the project	Options, Format, Style, Parameter, Configure, Import, Custom, Compatibility, Browser, Config, Client, Server, Module, Promise, Util, Class, Variables, Router
6	Release history	5,874 (13.35%)	✓	Release history of the project	Release History, ChangeLog, Change logs, Version history, Versions, Release Notes
7	Contribute	4,938 (11.23%)	✓	How to contribute to the project	How to Contribute, Contribution Guides, Development, Donations, CONTACT, Hacking
8	Test	4,374 (9.94%)		Test commands of the project	Run test, testing, To test, How to Test
9	Todo	4,293 (9.76%)		TODO list of the project	TODO, Todos, To-Do List, To-do soon, Task, In the future . . . , The future, Requirements, Coming soon!
10	Overview	4,219 (9.59%)	✓	Description of the project	Overview, Summary, Synopsis, Description, Introduction, Why?, What's this?, About The Name
11	Status	3,491 (7.94%)		Build status by continuous integration	Build Status, Current Status
12	Document	3,384 (7.69%)		Further Documentation for using the project	Documentation, Doc, Doe, Notes, Info, Information, TL;DR, Notice, Detail, See also
13	Author	2,607 (5.93%)	✓	Project author	Authors, About the Author, who am i?, Credits, Backers, Contributors, Other Contributors, Resources
14	Support	1,555 (3.53%)	✓	How to get support	Supports, FAQ?, Help, Troubleshooting, Questions
15	Feature	1,379 (3.13%)		Features list of the project	Key Features, Attributions
16	Relate	1,297 (2.95%)		Introduction of other projects related to the project	Related, Related Projects, Link, Inspirations, Alternatives, Source, Other libraries
17	Issue	1,039 (2.36%)		Issues	Issues, Known issues, Problems, Warning, Caveats, Bugs
18	Demo	839 (1.91%)		Example output of the project	Demo, Codepen demo, Example Output, Result
19	Purpose	821 (1.87%)		Purpose of the project	Purpose, Goals, Solution, Motivation, Background, Concepts, Key Ideas, Philosophy, Rationale
20	Refer	772 (1.75%)		References and Acknowledge	References, Thanks, Special Thanks, Acknowledgements

tem for users. The other content themes for users, "API" (24.27%) and the "Option" (23.78%) content theme explain the different list of functions for package usage. When there are many functions in a system, developers often note the feature list of the system with "Feature" content (3.13%). They may note "Support" (3.53%), "Demo" (1.91%), "Limit" (1.75%), and "Error" (1.04%) together as troubleshooting.

The lesser documented content themes are more related to contributors' information "Contribute", "Test", "Todo", "Issue", and "Roadmap". We suspected that such content themes may indicate that the packages are not fully mature and require more development to become stable for end users.

Observation 3 — "Overview", "Author" and "Support" are rarely noted in README files.

While "Overview", "Author" and "Support" are typically included in README as shown in Table 1, they account for less than 10% of the systems. We found that 9.59% for "Overview", 5.93% for "Author", and 3.53% for "Support" still exist in the README files. Furthermore, "Overview" is a more generic content theme while developers are less likely to include contributor content themes such as "Author" and "Support".

4.2 RQ2: Does the type of project affect how developers write their README file?

(1) Approach

To answer RQ2, we explore the relationship between the type of software project and its README content themes. In detail, as a case study, we compare two types of npm projects—*library-specific projects* (i.e., projects used by other applications as third-party libraries) and *application-specific projects* (i.e., projects used by end users as applications) [9]. Our approach consists of two steps:

(Step 1) Classification of README based on project type. As defined by Wittern et al. [9], we define two types of npm projects. The previous study found some keywords to describe for each type of packages. We can find the keywords in each project web site (e.g., The project "jquery"[†] tagged *"jquery", "javascript", "browser", and "library"*). The detail keywords are for GitHub-strong (i.e., identified by the keywords: "gruntplugin", "gulpplugin", "express", "react", "authentication") and (b) npm-strong (i.e., identified by the keywords: "util", "array", "buffer", "string", "file").

(Step 2): Identification of README content theme patterns. To identify common usage patterns between the two types of projects (i.e., GitHub-strong and npm-strong). Using our content theme results from RQ1, we then use the Association Rule Mining technique [11], [12] to identify common usage patterns.

Association Rule Mining is a method to extract a relationship between two or more items as an association rule from the combination of a large number of items. The association rule $\mathbb{R}$ is represented by a pre-condition, which is the README content (RC), and a post-condition (Pt) as follows: Let *RC* as a set of README content themes (i.e., "Install", "License", ...) and *Pt* refer to a set of project types (i.e., GitHub-strong or npm-strong).

$$\mathbb{R} = RC \Rightarrow Pt \tag{1}$$

To evaluate the extracted rules $\mathbb{R}$, we use the three metrics: support, confidence, and lift. We define the support as the proportion of rules where both pre-condition (RC) and post-condition (Pt) exist in all rules.

$$support(RC) = \frac{|RC \cap Pt|}{|Pt|} \tag{2}$$

The confidence metric is the proportion of rules which both the pre-condition (RC) and post-condition (Pt) exist in rules with the pre-condition (RC).

$$conf(\mathbb{R}) = \frac{support(RC \cup Pt)}{support(RC)} \tag{3}$$

Finally, Lift measures the magnification of the data in which the pre-condition (RC) and post-condition (Pt) exist in rules with the post-condition (Pt).

$$lift(\mathbb{R}) = \frac{conf(\mathbb{R})}{support(Pt)} \tag{4}$$

Our study implements association rule mining using the Orange [13] library in Python. The library uses an apriori algorithm, which is used to filter out the minor rules (i.e., using minimum support value (0.03), minimum confidence value (0.03), and minimum lift value (1)).

Table 6 shows the identified README files (Step 1). In fact, we extracted 2,788 GitHub-strong type packages and 1,870 npm-strong type packages from our dataset. During the analysis, we discarded the keyword "gruntplugin" as gruntplugin related projects use different README format. We also ignore systems which include keywords from both project types.

We report our README content theme patterns in two forms. Table 7 shows the extracted 36 rules which the target README files frequently use sorted by lift value. The second representation is a graph-based visualization of the generated rules.

Figure 3 shows how our generated rule is translated into a graph representation. In this example, each of the incoming edges (i.e., "Usage", "Install", "License", "Test") represents each of the precondition content themes, while the outgoing edge is the postcondition (i.e., npm-strong) The color of the node represents the lift metric, and as shown in the example, the color shows 1.49 lift. Our assumption is that the colors will show whether or not the content theme is related to either "GitHub-strong" or "npm-strong".

Figure 4 shows the generated graph (i.e., displayed using the Fruchterman Reingold algorithm) for the 36 rules generated in Table 7. The color of the node follows the lift score from a light color (low) to a strong color (high).

Table 6 GitHub-strong type packages and NPM strong packages.

GitHub-strong type packages	
Filtered Projects	2,788 packages
Content Theme	2,254 kinds of headline variations

npm-strong type packages	
Filtered Packages	1,870 packages
Content Theme	1,567 kinds of headline variations

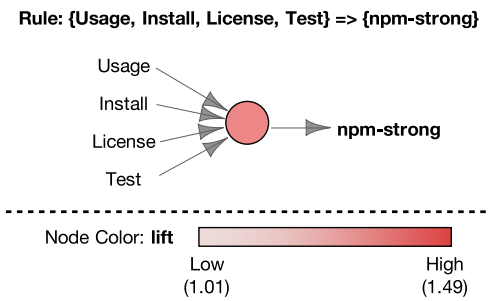

Fig. 3 An example of the content theme pattern rule represented as a directed graph, with nodes and edges. Note that the color indicates the lift metric.

[†]jquery https://www.npmjs.com/package/jquery

Table 7 Top 36 Association Rules for GitHub-strong vs. npm-strong type packages .

Id	Content Themes	⇒	Project type	support	confidence	lift
1	{Usage,Install,License,Test}	⇒	{npm-strong}	0.04	0.60	1.49
2	{Install,License,Status}	⇒	{npm-strong}	0.03	0.59	1.48
3	{Install,License,Test}	⇒	{npm-strong}	0.05	0.59	1.47
4	{License,Status}	⇒	{npm-strong}	0.04	0.58	1.44
5	{Usage,License,Test}	⇒	{npm-strong}	0.04	0.58	1.43
6	{License,Test}	⇒	{npm-strong}	0.05	0.57	1.42
7	{Install,Status}	⇒	{npm-strong}	0.04	0.56	1.41
8	{Usage,Install,Test}	⇒	{npm-strong}	0.05	0.56	1.39
9	{Usage,Status}	⇒	{npm-strong}	0.04	0.54	1.35
10	{Install,Test}	⇒	{npm-strong}	0.06	0.54	1.35
11	{Install,License,API}	⇒	{npm-strong}	0.06	0.54	1.34
12	{Usage,Test}	⇒	{npm-strong}	0.06	0.54	1.34
13	{Status}	⇒	{npm-strong}	0.05	0.53	1.33
14	{Test}	⇒	{npm-strong}	0.07	0.53	1.31
15	{Usage,Option}	⇒	{GitHub-strong}	0.14	0.76	1.27
16	{License,Option}	⇒	{GitHub-strong}	0.08	0.75	1.25
17	{Install,API}	⇒	{npm-strong}	0.09	0.50	1.25
18	{Install,Option}	⇒	{GitHub-strong}	0.12	0.74	1.24
19	{Option}	⇒	{GitHub-strong}	0.19	0.73	1.21
20	{License,API}	⇒	{npm-strong}	0.07	0.47	1.18
21	{Install,License}	⇒	{npm-strong}	0.16	0.47	1.17
22	{Usage,Install,Releas History}	⇒	{npm-strong}	0.03	0.46	1.16
23	{Install,Author}	⇒	{npm-strong}	0.03	0.46	1.16
24	{Install,Releas History}	⇒	{npm-strong}	0.04	0.46	1.14
25	{Todo}	⇒	{GitHub-strong}	0.04	0.68	1.13
26	{API}	⇒	{npm-strong}	0.11	0.45	1.13
27	{Usage,Overview}	⇒	{GitHub-strong}	0.04	0.67	1.11
28	{License}	⇒	{npm-strong}	0.20	0.44	1.10
29	{Install,Overview}	⇒	{GitHub-strong}	0.04	0.65	1.09
30	{Overview}	⇒	{GitHub-strong}	0.05	0.65	1.08
31	{Author}	⇒	{npm-strong}	0.04	0.43	1.07
32	{Install}	⇒	{npm-strong}	0.25	0.42	1.05
33	{Releas History}	⇒	{npm-strong}	0.05	0.42	1.04
34	{Contribute}	⇒	{npm-strong}	0.04	0.41	1.03
35	{Usage}	⇒	{GitHub-strong}	0.41	0.61	1.02
36	{Document}	⇒	{GitHub-strong}	0.05	0.61	1.01

Fig. 4 36 README content theme rules generated as a directed graph.

(2) **Result**

Observation 1 — "Install" and "License" content themes are likely to be important for a npm-strong type package.

While RQ_1 shows that a README file typically includes "Usage", "Install" and "License", Fig. 4 provides evidence that the "License" content theme is closely related to the npm-strong type of packages. Correspondingly, Table 7 shows that many npm packages with higher lift and higher confidence scores have some of the same rules as "License". From this result, we believe that it is important for core utility packages to share the license because it is frequently reused by the other systems. Additionally, the graph shows that the "Status" and "Test" content themes are more closely associated closely to npm-strong types of packages. Furthermore, we suspect that the developers of these packages may be expected to be adapted correctly to the 'test' and the 'build' states.

Observation 2 — The "Option" content theme is likely to be an important content theme for GitHub-strong type packages. Figure 4 shows that "Option" is closely related to the GitHub-strong type packages. Table 7 shows that many of the GitHub-strong type packages have higher lift

and higher confidence scores with rules associated with the "Option" content theme. Furthermore, we suspect that the "Option" content themes are more important for the end-user package, as end user packages are more likely to have more lists of product options than npm-strong type (i.e., utility) packages.

5. Summary of Results

In order to aid developers faced with documentation issues, we conducted an empirical study to understand the written content themes of the README file. The co-founder of GitHub Tom Preston-Werner, even discussed the importance of the README file, coining Readme Driven Development (RDD)[†] as an important subset of Document Driven Development. We learned some valuable lessons along the way:

- *Lesson 1: Although a README file contains numerous variations, we built a taxonomy of 20 README content themes* - Surprisingly, from over 30,000 content theme variations, we were able to build a taxonomy of 20 headline content themes, which are used by more than 1% of packagess. We conjecture that the content themes may reveal insights such as project practices or may be an indicator of changes in the project.
- *Lesson 2: Content themes "Usage", "Install", and "License" are common README content themes* - "Usage", "Install", and "License" are typically included in the README. Furthermore, less apparent README content themes include "API", "Test", and "Todo", used in 10%–24% of packages. Such information may be important, especially for novice developers.
- *Lesson 3: Especially for npm packages, the study shows that "Install" and "License" are likely content themes for library-specific packages, while the "Option" content theme is more common for application-specific packages.*
  We found some specific README content themes according to the type of projects. We found "Install" (40% packages) and "License" (20% packages) are common for npm libraries, while nodeJS application packages included the option content themes (10% packages). Such information may be important for developers, especially for novice developers.

6. Threats to Validity

External validity - refers to the generalization concerns of the study to other software systems including some package ecosystem such as Java library and Ruby RubyGems. This study found some specific results for the npm package ecosystem. For example, while "Install" is reported as one a major content theme from our findings and is used by 59.43% of systems, we carefully restrict these findings to the npm package ecosystem because other systems may depict different patterns and tendencies for a README file. This might be an interesting future avenues for research.

Internal validity - refers to the concerns that are internal to this study. In this study, we found two main internal threats that could affect our results. First is the preprocessing of the dataset. In RQ_1, we classified 30,939 content themes into the 20 most frequent content themes by merging the more frequent contents and filtering out the less common content themes. The manual merging of content themes in RQ_1 was conducted through a reached consensus among authors. However, we followed a strict iterative process and are confident of the results. The second threat is related to the content themes of the README files (i.e., RQ_2). As shown in our results, not every README file will include key content themes. For example, some projects have separate meta-files for licenses; thus, the content theme for licenses may not exist in the README file. For future work, it will be interesting to investigate all meta-files to understand how developers maintain and keep all files.

Construct validity - refers to the concerns of the result. We found one threat that related to the extraction of content themes from the README file. This study used the Markdown Format to extract the headline levels 1 and 2 (i.e., *h1* and *h2*). There may, however, be cases where the project is using level 3 (i.e., *h3*) to write major content themes. Nonetheless, we are confident of the results and of our extraction approach.

7. Conclusions and Future Work

In this paper, we investigated content themes of the README file. Although we found that the README file contains numerous ambiguous naming variations, we were able to summarize and build a taxonomy of 20 README content themes used by more than 1% of packages. The results show that README files contain common content themes such as "Usage", "Install", and "License", as outlined in known guidelines. Furthermore, we found that "Install" and "License" are likely content themes for library-specific packages, while the "Option" content theme is more common for application-specific packages. Finally, we showed that packages rarely remove README content themes.

As future work, we would like to extend our project types and techniques to provide more comprehensive guidelines for writing a good README file. We also believe that further understanding of README will assist both developers and their end users in keeping up with ongoing changes in a project.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 16K16037, 18KT0013, 18H04094 and 17H00731.

[†]Readme Driven Development: http://tom.preston-werner.com/2010/08/23/readme-driven-development.html

References

[1] J. Coelho and M.T. Valente, "Why modern open source projects

fail," Proc. 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '17), pp.186–196, 2017.
[2] T.C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," IEEE Softw., vol.20, no.6, pp.35–39, 2003.
[3] S.L. Abebe, N. Ali, and A.E. Hassan, "An empirical study of software release notes," Empirical Software Engineering, vol.21, no.3, pp.1107–1142, 2016.
[4] L. Moreno, G. Bavota, M.D. Penta, R. Oliveto, A. Marcus, and G. Canfora, "ARENA: An approach for the automated generation of release notes," IEEE Trans. Softw. Eng., vol.43, no.2, pp.106–127, 2017.
[5] J. Kim, S. Lee, S.-W. Hwang, and S. Kim, "Enriching documents with examples: A corpus mining approach," ACM Trans. Information Systems, vol.31, no.1, pp.1–27, 2013.
[6] Y. Zhou, R. Gu, T. Chen, Z. Huang, S. Panichella, and H. Gall, "Analyzing APIs documentation and code to detect directive defects," Proc. 39th International Conference on Software Engineering (ICSE '17), pp.27–37, 2017.
[7] F. Hassan and X. Wang, "Mining readme files to support automatic building of Java projects in software repositories," Proc. 39th International Conference on Software Engineering Companion (ICSE '17), pp.277–279, 2017.
[8] Y. Zhang, D. Lo, P.S. Kochhar, X. Xia, Q. Li, and J. Sun, "Detecting similar repositories on GitHub," Proc. IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER '17), pp.13–23, 2017.
[9] E. Wittern, P. Suter, and S. Rajagopalan, "A look at the dynamics of the JavaScript package ecosystem," Proc. 13th International Conference on Mining Software Repositories (MSR '16), pp.351–361, 2016.
[10] F.N.A.A. Omran and C. Treude, "Choosing an NLP library for analyzing software documentation: A systematic literature review and a series of experiments," Proc. 14th International Conference on Mining Software Repositories (MSR '17), pp.187–197, 2017.
[11] C. Zhang and S. Zhang, Association rule mining: Models and algorithms, Springer-Verlag, 2002.
[12] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," Proc. International Conference on Management of Data (SIGMOD '00), pp.1–12, 2000.
[13] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan, "Orange: Data mining toolbox in Python," Journal of Machine Learning Research, vol.14, pp.2349–2353, 2013.

Shohei Ikeda received the B.E. degree from the National Institute of Technology, Nara College, Japan in 2016. He is currently a Master's Degree student at the Nara Institute of Science and Technology. His research interests includes software engineering.

Akinori Ihara received the B.E. degree in Science and Technology from Ryukoku University, Japan in 2007, and the M.E. degree (2009) and D.E. degree (2012) in Information Science from the Nara Institute of Science and Technology, Japan. He is currently a lecturer at Wakayama University, Japan from 2018. And, he was an Assistant Professor at Nara Institute of Science and Technology from 2012. His research interests include the quantitative evaluation of open source software development process. He is a member of the IEEE and IPSJ.

Raula Gaikovina Kula is currently a Specially Appointed Assistant Professor at Nara Institute of Science and Technology. In 2013, he graduated with a Ph.D. from Nara Institute of Science and Technology, Japan. He is currently an active member of the IEEE Computer Society and ACM. His research interests include repository mining, code review, software libraries and visualizations.

Kenichi Matsumoto received the B.E., M.E., and Ph.D. degrees in Engineering from Osaka University, Japan, in 1985, 1987, 1990, respectively. He is currently a professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. His research interests include software measurement and software process. He is a senior member of the IEEE and a member of the IPSJ and SPM.