

**Doctoral Dissertation**

**Understanding Third-party Library Discussions  
Through Question-and-Answer Sites**

**Syful Islam**

Program of Information Science and Engineering  
Graduate School of Science and Technology  
Nara Institute of Science and Technology

Supervisor: Kenichi Matsumoto  
Software Engineering Lab. (Division of Information Science)

Submitted on March 15, 2022

A Doctoral Dissertation  
submitted to Graduate School of Science and Technology,  
Nara Institute of Science and Technology  
in partial fulfillment of the requirements for the degree of  
Doctor of Engineering

Syful Islam

Thesis Committee:

Supervisor    Kenichi Matsumoto  
                  (Professor, Division of Information Science)  
                  Hajimu Iida  
                  (Professor, Division of Information Science)  
                  Takashi Ishio  
                  (Associate Professor, Division of Information Science)  
                  Raula Gaikovina Kula  
                  (Assistant Professor, Division of Information Science)  
                  Mohammed Humayun Kabir  
                  (Professor, Noakhali Science and Technology University)

# Understanding Third-party Library Discussions Through Question-and-Answer Sites\*

Syful Islam

## Abstract

Third-party libraries have become an integral part of modern software development, as developers have access to many useful libraries through the software packaging ecosystems. Managing third-party libraries is one of the major challenges of the software ecosystem due to its high degree of interdependence. In addition, it is difficult for developers to understand the challenges and information needs of using third-party libraries from different ecosystems during the software development process. Question-and-answer sites are a popular source of knowledge for communicating developer challenges.

This thesis investigates third-party package (i.e., library) related discussions through a popular question-and-answer site such as Stack Overflow to understand the developer's experience (i.e., the challenges and information needs) of using third-party packages from different software packaging ecosystems. The main contributions of this thesis are divided into two parts. In the first part, developer's experience in package management is investigated, and found that discussion topics vary by packaging ecosystem. The results also indicate that certain features (e.g., dependency tree, environment, etc.) of package management correlate with the developer's experience. In the second part of the thesis, the investigation into package usage reveals that developers encounter different types of errors while maintaining third-party packages. Furthermore, popular packages are less likely to be discussed on Stack Overflow, indicating fewer issues relating to their usage. Finally, analyzing accepted answers shows that the answers, which include usage examples and execution commands, help resolve errors related to package usage.

---

\*Doctoral Dissertation, Graduate School of Science and Technology, Nara Institute of Science and Technology, March 15, 2022.

In summary, the results of this thesis highlight the challenges and information needs associated with using third-party packages from software packaging ecosystem for developers, package manager designers and researchers.

**Keywords:**

Third-party Library, Software Ecosystem, Package Management, Package Usage, Question-and-Answer site, Developers Experience

## Acknowledgements

First of all, I would like to thank **Almighty Allah** for giving me the opportunity, determination and strength in my PhD study. His constant grace and mercy has been with me throughout my life.

Secondly, I would like to express my sincere gratitude to **Professor Michiko Inoue** for accepting me as a PhD student of NAIST. I sincerely thank her for everything she had done for me. Thirdly, sincere gratitude to my PhD supervisor **Professor Kenichi Matsumoto** for giving me opportunities to study in his laboratory as a PhD student. His guidance and motivation helped me a lot in my research and writing of this thesis. This thesis would never have been accomplished without his generous support. I believe, my supervisor did for me what the best supervisors always do for his/her students to make them successful in life.

To my thesis committee, I would like to thank Professor Hajimu Iida, Associate Professor Takashi Ishio, and Professor Mohammed Humayun Kabir for their valuable comments and discussion to improve my thesis.

My special thanks goes to Assistant Professor Raula Gaikovina Kula for his continual support and guidance. His guidance and support from the initial to final level helped to develop a good understanding in my research area. He has literally taught me how to do research and motivated with great insights and innovative ideas. He has guided me the ways to visualize the problem with all possible angles and to get a deep insight into the topic. His presence always helped me to work harder and be simple.

Also special thanks goes to Senior Lecturer Christoph Treude (University of Melbourne), who motivated me with great insights and innovative ideas during my PhD study.

I would like to thank my lab mates in software engineering lab for great help, support, and encouragement from the beginning to the end.

Last but not least, I would like to thank my family for their love, and encouragement to study until now. Without them, I would not have a chance to pursue my dream in Japan. Finally, I would like to express my most sincere appreciation to MEXT and NAIST international affair division for all kind of supports.

## List of Publications

- **An Exploration of npm Package Co-Usage Examples from Stack Overflow: A Case Study**  
Syful Islam, Dong Wang, Raula Gaikovina Kula, Takashi Ishio, and Kenichi Matsumoto. IEICE Transactions on Information and Systems, Special Section on Empirical Software Engineering, Volume and Number: Vol.E105-D ,No.1, pp.11-18, Jan. 2022.
- **Contrasting Third-Party Package Management User Experience**  
Syful Islam, Raula Gaikovina Kula, Christoph Treude, Bodin Chinthanet, Takashi Ishio, Kenichi Matsumoto. International Conference on Software Maintenance and Evolution , IEEE, pp. 664-668, 27 Sep. 2021.
- **Choice Matters: Contrasting Package Manager User Experience**  
Raula Gaikovina Kula, Syful Islam, Bodin Chinthanet, Christoph Treude, Takashi Ishio, Kenichi Matsumoto. 4th International Workshop on Software Health in Projects, Ecosystems and Communities, May 29th, 2021.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of publications</b>	<b>iv</b>
<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1 Scope of this Thesis . . . . .	2
2 Contributions . . . . .	4
2.1 Package management . . . . .	4
2.2 Package usage . . . . .	5
3 Organization of Thesis . . . . .	6
<b>2 Background</b>	<b>8</b>
1 Third-party Package Usage in Modern Software Development . . . . .	8
1.1 Usefulness of third-party packages . . . . .	9
1.2 Drawback of third-party packages . . . . .	10
2 Third-party Package Management . . . . .	11
2.1 Basic Functions of Package Manager . . . . .	12
3 Software Packaging Ecosystem . . . . .	12

3.1	Evolution of Software Packaging Ecosystems . . . . .	12
4	Developers Discussion on Question-and Answering Site . . . . .	16
4.1	Stack Overflow: A Popular Q&A Site Among Developers . . . . .	16
	Question . . . . .	17
	Answer . . . . .	17
	Comments . . . . .	19
5	Definition of Developers Experience . . . . .	19
6	Related Works . . . . .	19
<b>3</b>	<b>Package Management</b>	<b>24</b>
1	Introduction . . . . .	24
2	Contrasting Developers Experience in Package Management . . . . .	25
2.1	Selecting package ecosystems and their features . . . . .	26
2.2	Building third-party package related question-and-answer dataset . . . . .	27
2.3	Question Topic Modeling . . . . .	28
2.4	Contrasting developer discussion topics relating to different ecosystems . . . . .	34
	A. Contrasts in Responses: Popularity and Difficult . . . . .	34
	B. Contrasts in Features: Topics and Features . . . . .	39
3	Investigating Root Cause of Package Management Discussion and Their Kinds of Questions . . . . .	43
3.1	What kind of questions do developers ask about third-party package management? . . . . .	44
3.2	What are the underlying causes of questions related to third-party package management? . . . . .	47
4	Implications . . . . .	51
5	Threats to Validity . . . . .	52
6	Summary and Future Works . . . . .	53
<b>4</b>	<b>Package Usage</b>	<b>55</b>
1	Introduction . . . . .	55
2	Motivating Example . . . . .	56
3	Data Preparation . . . . .	59



4	Data Analysis . . . . .	60
4.1	Approach for RQ <sub>1</sub> : Package usage issues . . . . .	60
4.2	Approach for RQ <sub>2</sub> : Developers practices to solve package usage issues . . . . .	61
5	Results . . . . .	63
5.1	Answering RQ <sub>1</sub> : Package usage issues . . . . .	63
5.2	Answering RQ <sub>2</sub> : Developers practices to solve package usage issues . . . . .	68
6	Discussion . . . . .	68
7	Threats to validity . . . . .	69
8	Summary and Future Works . . . . .	70
<b>5</b>	<b>Conclusion</b>	<b>72</b>
1	Implications and Suggestions . . . . .	73
2	Opportunities for Future Research . . . . .	74

# List of Figures

1.1	Area of thesis. . . . .	3
1.2	An overview of the structure of this thesis. . . . .	6
2.1	The package usage scenario to develop pandas package and their different dependency types. According to the figure, pandas package has three main dependencies (i.e., numpy, python-dateutil, and pytz), and the rest dependencies are optional. . . . .	9
2.2	Typical workflow of third-party package management. It illustrates the workflow of python package manager pip to install the pandas packages from PyPI package repository. . . . .	11
2.3	Overview of the evolution of thirteen third-party package ecosystems of top programming languages according to TIOBE index <a href="https://www.tiobe.com/tiobe-index/">https://www.tiobe.com/tiobe-index/</a> . . . . .	13
2.4	Example of Stack Overflow question (Id: 898782) where a developer ask for the solution of CPAN dependency management. . . . .	17
2.5	Example of Stack Overflow answers of a question (Id: 898782) where other Stack Overflow users propose solution for the CPAN dependency management. . . . .	18
3.1	An overview of contrasting developers experience in third-party package management on 13 selected package ecosystems from top 20 programming languages. . . . .	26
3.2	Distribution of third-party package related discussion topics and their higher level theme (in percentage). . . . .	31

3.3	Package ecosystem vs. topic, where developers of different package ecosystems ask different questions. . . . .	41
3.4	Package ecosystem vs. topic themes, where most developers of package ecosystems ask questions on package management except Go and Meteor . . . . .	41
3.5	Language feature heatmap shows that different package ecosystems in different languages provide different developers experiences. . .	42
3.6	Environment feature heatmap shows that different package ecosystems in different environments provide different developers experiences. . . . .	42
3.7	Dependency tree feature heatmap shows that dependency tree correlates with developers experiences (i.e., package ecosystems with a nested dependency face package usage issues, while flat dependency trees face questions related to dependencies) . . . . .	43
3.8	Percentage of third-party package posts by question coding compared to Treude et al. [91]. Result shows that ‘How-to’ and ‘Error’ messages are the most dominant questions asked. . . . .	46
3.9	Post (Id: 50262939) that discuss issues related to specific migration	48
3.10	Post (Id: 4811870) that discuss issues related to package management tool usage . . . . .	49
3.11	Post (Id: 30571) that discuss issues related to general idea of dependency practice. . . . .	49
3.12	Percentage of third-party package posts by their underlying cause. I find that ‘ <i>Package management tool usage</i> ’ is the most dominant underlying cause. . . . .	50
4.1	A motivating example of npm package co-usage in Stack Overflow. The example shows that a developer encounters a issue when installing all node_modules, due to two dependent packages <b>babel-loader</b> and <b>webpack</b> . . . . .	57
4.2	An overview of the methodology of our study. . . . .	58
4.3	Word cloud generated from Stack Overflow npm posts title that contains package usage information. The word cloud shows that npm posts are primarily related to various types of errors. . . . .	62

4.4	An example that motivates to classify developers response. In the answer I observe that, it contains usage examples, execute command, and step by step instruction. . . . .	64
4.5	Analysis of accepted answers posted in response to questions that include package usage information. Result shows that 37.76% accepted answers contain <code>usage example</code> followed by <code>execute command</code> 19.58%. . . . .	69

# List of Tables

3.1	Summary of each package ecosystem and their features. . . . .	27
3.2	List of the 28 manually validated tags discovered from Step 2 of the data preparation. . . . .	29
3.3	Topics and their top ten keywords extracted from third-party package related question posts. The topics are categorized into three themes. . . . .	35
3.4	Popularity of third-party package discussion topics. All the metrics are represented in median value. The median value of favourite count for all topics are zero. Result shows that error and testing related issues are most popular among developers. . . . .	37
3.5	Difficulty of third-party package discussion topics. The answer count metric is represented in median value and the accepted answer count and PD score are represented in percentage (%). Result shows that error topic questions are most difficult to answer while package usage issues are easiest to answer by other stack overflow users. . . . .	38
4.1	Summary of dataset used in the study. . . . .	59
4.2	Top-15 npm packages extracted from Stack Overflow posts with their proportion and rank in the latest npm projects. Result shows that Only three out of top-10 npm packages are mostly discussed in Stack Overflow. . . . .	65

4.3	Top-15 package usage extracted from the latest npm projects with their proportion and rank in the Stack Overflow posts. The top package usage patterns from npm projects shows that application developers top usage packages are different from Stack Overflow. .	66
4.4	Top-15 package co-usage extracted from Stack Overflow posts except <b>angular</b> since rest of the top co-usage are related to <b>angular</b> . The top co-usage patterns and their rank in Stack Overflow indicate that developers discuss most package dependency issues related to <b>angular</b> followed by ( <code>'typescript'</code> , <code>'zone.js'</code> ). . . .	67

# 1 | Introduction

Third-party libraries have become an integral part of modern software development, as many useful libraries for different programming languages are readily available to developers through the library distribution systems. These library distribution systems are popularly known as software packaging ecosystem, consisting of a large set of library components developed, shared, and evolved on common technology platforms to deliver numerous software services or solutions [25, 51, 57]. Due to the increasing popularity of third-party library usage, the software ecosystems tend to grow rapidly containing millions of libraries for developers. For instance, the `npm` packaging ecosystem has provided over 800,000 free and reusable software packages and is trusted by over 11 million developers around the world<sup>1</sup>. For the rest of this thesis, the term `package` is used instead of `library` because the software ecosystem introduces the library as a package containing code module binaries, configuration files, and their dependencies.

Managing third-party packages is one of the major challenges in software ecosystems, since they are highly interdependent [4, 17, 18]. As the number of dependencies grows (i.e., forming a large tree of interdependent packages) within the application, so do the chances of incompatibility between dependencies. Several studies reported the negative impact of third-party package overuse including increased maintenance cost [72, 87], security vulnerability and backward incompatibility issues [24, 29, 40, 71]. For example, in a recent incident of a `npm` package called `left-pad`, which was used by Babel, caused interruptions to the popularly used internet websites, e.g., Facebook and Netflix<sup>2</sup>. To address such

---

<sup>1</sup><https://www.npmjs.com/>

<sup>2</sup><https://www.sciencealert.com/how-a-programmer-almost-broke-the-internet-by-deleting-11-lines-of-code>

problems, the package managers act as intermediary brokers between an application and a package dependency to ensure that a verified package is correctly installed, configured, or removed from an application. They solve the ‘dependency hell’<sup>3</sup> dilemma, i.e., to avoid compatibility and build issues that arise when an application adopts numerous dependencies.

Diversity in technology stacks and programming languages has led to a variety of managers that cover different software packaging ecosystems. For instance, npm brokers packages that run in the Node.js environment and are written in JavaScript, while the PyPI package ecosystem is built specifically to handle Python package dependencies. Recent studies have investigated ecosystem tooling policies, dependency management from the perspectives of updates (i.e., update an existing dependency to a more recent version) and migration (i.e., replace, remove, or add a new dependency) [17, 18, 27, 28, 30, 41, 45]. For example, Bogart et al. [17, 18], performed multiple case study on a set of package ecosystems with different tooling policies. They found that “developers practices differ significantly between ecosystems and all ecosystems share values such as stability and compatibility, but differ in other values”.

Selecting an appropriate software packaging ecosystem is crucial for developers, specially when building mobile and web applications. In addition, it is difficult for developers to understand the challenges and information needs of using third-party packages from different ecosystems. The purpose of this thesis is to investigate the developers experience (i.e., the challenges and information needs) in different software packaging ecosystems on (1) package management and (2) package usage. To the best of my knowledge, no research has been done on the developer’s experience in using third-party packages from different software packaging ecosystems.

## 1 Scope of this Thesis

This section presents the scope of this thesis in terms of developers experience as illustrated in Figure 1.1.

---

<sup>3</sup>A term made popular by this blog <https://web.archive.org/web/20150708101023/http://archive09.linux.com/feature/155922>



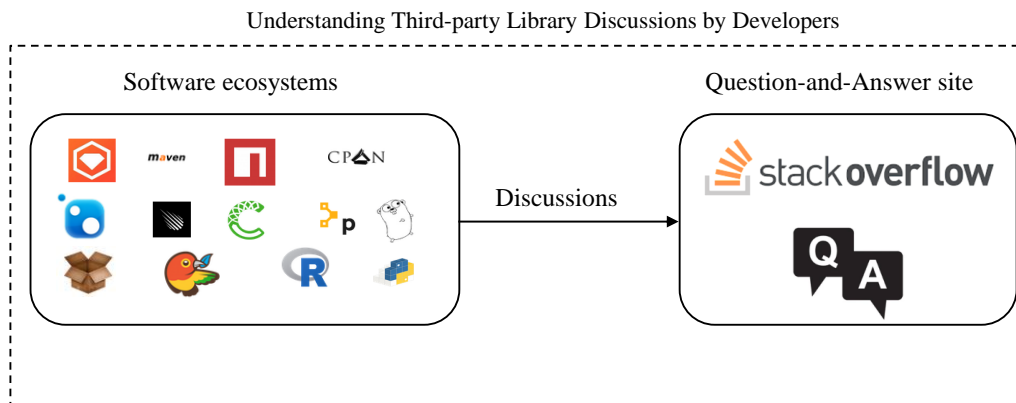


Figure 1.1: Area of thesis.

Community based Question-and-answering (Q&A) sites such as Stack Overflow has gained huge popularity as crowd-sourced knowledge for software developers to solicit solutions to their challenges and information needs. Previous studies by Gupta and Reddy [36], Storey et al. [83] reported that querying information needs from Stack Overflow is a regular activity by many software developers. A systematic study by Meldrum et al. [62] has shown that Stack Overflow attracted increasing research interest over the years, with topics relating to community dynamics, technical issues of developers, and human factors.

In this thesis, I hypothesized that **“Question-and-answering discussions can provide valuable insights into the challenges and information needs associated with using third-party package from software packaging ecosystems.”**

To validate my thesis statement, I utilized Stack Overflow as shown in Figure 1.1 to understand third-party package related discussions by developers. As far as I know, there was no prior work that conducted study on third-party package discussions from Stack Overflow. To investigate developers experience in different third-party package ecosystems, I collected relevant Q&A posts from Stack Overflow and perform a series of empirical studies. I believe that this thesis sheds the spot light on the challenges and information needs associated with using third-party packages from software packaging ecosystem for developers, package manager designers and researchers.

## 2 Contributions

To better understand the developer experience with using third-party packages from different software packaging ecosystems, this thesis has conducted a series of empirical studies. The contributions of this thesis are mainly divided into two fold; developers experience in (i) package management, and (ii) package usage. In the following, I have summarized the main results.

### 2.1 Package management

This chapter investigates developers experience on third-party package management in different software packaging ecosystems. To accomplish the goal, this chapter departs with two consecutive studies. At first, I performed a quantitative study on 497,249 Stack Overflow Q&A posts from 13 packaging ecosystems to contrast developers experience of package management. Second, I conducted a qualitative study on 1131 third-party package question posts from 3 packaging ecosystems to get deeper understanding of the questions asking and their underlying causes.

Among the main findings, the quantitative study identifies 10 topics that developers are discussing on third-party packages which are further categorized under three major themes: (i) Package management, (ii) Input/Output, and (iii) Package. Contrasting developers response in terms of topic popularity and difficulty, I find that third-party package posts that relate to error and testing topics are most popular and also most difficult for developers to solve. In addition, contrasting developers experience based on package management features, I observe that developers experience is different, depending on the choice of packaging ecosystem. After combining the results of quantitative study, I speculate that developers using Go and Meteor package ecosystems have relatively easier time finding answers to their questions on Stack Overflow compared to developers using other package ecosystems.

Again, the qualitative study identifies that third-party package related questions arise due to lack of instructions (i.e., around 31 to 42% of question samples) and error messages (i.e., around 27% to 37% of questions). Further investigation on third-party package posts shows that the underlying causes can be catego-

rized into package management tool usage, general dependency practices, specific migration, and others. An analysis of the underlying causes in 3 packaging ecosystems reveals that most questions related to third-party packages by developers are due to lack of knowledge on package management tool usage, rather than specific migration.

## 2.2 Package usage

In this chapter, an exploratory study on package usage information from Stack Overflow is performed in term of co-usage relationship. The rationale behind refining the co-usage relationship is to study problems caused by npm packages. In particular we investigate (i) whether we can detect package usage (i.e., co-usage) information from Stack Overflow and (ii) what the developers are looking for to solve problems related to the package. To address these, we study over 2,100 Stack Overflow Q&A posts and matched them to 217,934 npm library packages. I reveal the following valuable lessons along the way:

- *Lesson 1:* I find that only three out of the top ten of the most used npm libraries are mentioned in Stack Overflow. The top-3 discussed npm libraries are `react`, `typescript`, and `webpack`. Again, the top-5 libraries that are less frequently discussed in Stack Overflow are `mocha`, `eslint`, `chai`, `babel-core`, and `lodash`. One possible reason is that, well-known libraries are well documented and may have their own forum, chat tools, etc. For this reason, there is no need to discuss them in Stack Overflow. Furthermore, we find that 87.95% of package co-usage mined from Stack Overflow exist in the latest npm package release.
- *Lesson 2:* Developers post answers provided with usage example or execute command. Results do indicate the potential for a recommendation system, especially with the available execute commands and examples. Although Stack Overflow has been a useful resource for finding answers to questions, I find that popular and highly used libraries are not discussed as often. However, the accepted answers may prove useful, as I believe that the usage examples and executable commands could be reused for tool support.

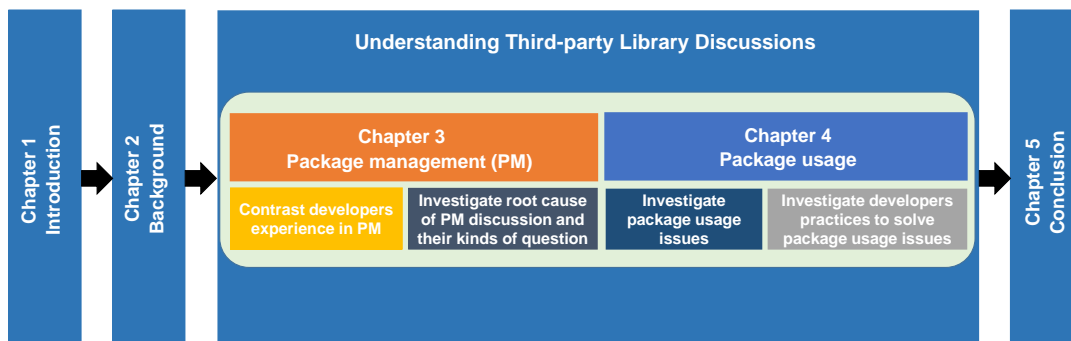


Figure 1.2: An overview of the structure of this thesis.

### 3 Organization of Thesis

In this section, I provide an organization of this thesis. Figure 1.2 illustrates the organization of the thesis. The details of this thesis is structured as follows:

Chapter 2 presents the background of this thesis which includes details of package ecosystem, Q&A website, the key term developers experience, and related works. In detail, Section 1 defines third-party package usage along with its usefulness and limitations. Section 2 defines package management system and its functions in detail. Section 3 defines third-party package ecosystem and describe their evolution. Section 4 introduces the detail of a question-answering discussion by taking Stack Overflow as a case study. Section 5 defines the key term ‘*Developers Experience*’ which is used throughout this thesis. Finally, Section 6 discusses the related works of this thesis.

Chapter 3 investigates developers experience on third-party package management in different software packaging ecosystems. Section 2, presents quantitative study to contrast developers experience in package management. Section 3 presents the qualitative study to investigate the root causes of package management discussion and their kinds of questions. Section 4 discusses the implications of this chapter. Section 5 presents the possible threats to validity of results. Finally, in Section 6, I summarize the results.

Chapter 4, investigates the package usage information from Stack Overflow in term of co-usage relationship. In detail, Section 2 presents motivating example and research questions. Section 3 describes the data preparation. Section 4

presents the analysis approach. Section 5 reports the results for each research question. Section 6, discusses the implications from this study. Section 7 discloses the threats to validity of our study. Finally, I summarize this chapter in section 8.

Finally, Chapter 5 concludes the thesis.

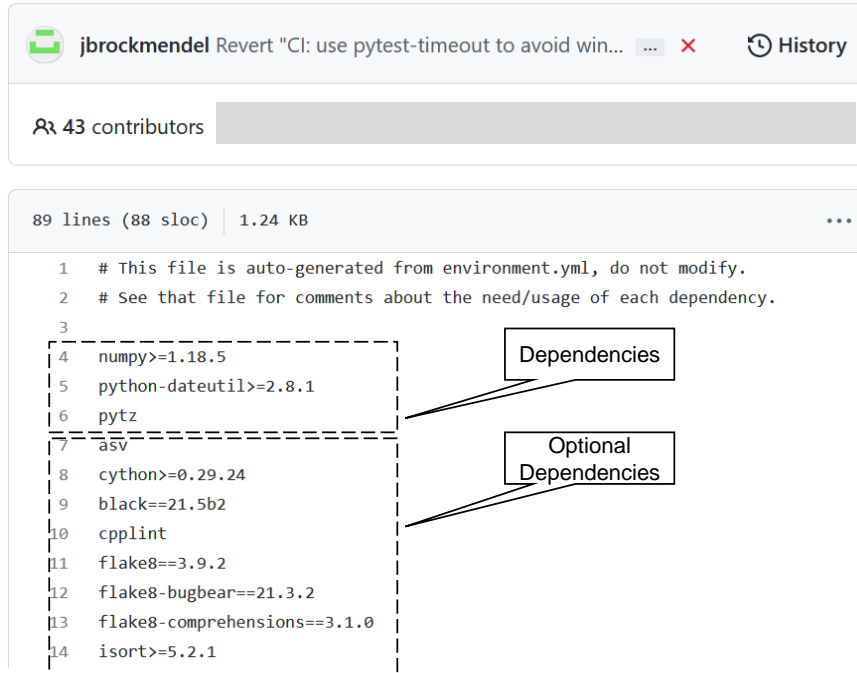
## 2 | Background

The purpose of this Chapter is to describe background of this thesis which includes details of package ecosystem, Q&A website, the key term developers experience, and related works. In detail, Section 1 defines third-party package usage along with its usefulness and limitations. Section 2 defines package management system and its functions in detail. Section 3 defines third-party package ecosystem and describe their evolution. Section 4 introduces the detail of a question-answering discussion by taking Stack Overflow as a case study. Section 5 defines the key term ‘*Developers Experience*’ which is used throughout this thesis. Finally, Section 6 discusses the related works of this thesis.

### 1 Third-party Package Usage in Modern Software Development

Third-party packages are reusable software components developed to be freely distributed or sold by entities other than original vendor of the development platform. These packages typically contain information about code module binaries, configuration files, and their dependencies [76]. The usage of third-party packages have become an integral part in modern software development as it facilitates to integrate a functionality without writing code from scratch [14, 47, 69, 74, 75, 88]. Figure 2.1 shows the package usage scenario to develop pandas software package. According to the figure, pandas package has three main dependencies (i.e., numpy, python-dateutil, and pytz), and the rest dependencies are optional. Below, I explain the usefulness and drawback of third-party package usage in modern software development.

pandas / requirements-dev.txt



```
1 # This file is auto-generated from environment.yml, do not modify.
2 # See that file for comments about the need/usage of each dependency.
3
4 numpy>=1.18.5
5 python-dateutil>=2.8.1
6 pytz
7 ---
8 asv
9 cython>=0.29.24
10 black==21.5b2
11 cpplint
12 flake8==3.9.2
13 flake8-bugbear==21.3.2
14 flake8-comprehensions==3.1.0
15 isort>=5.2.1
```

The screenshot shows a code editor window titled 'jbrockmendel Revert "CI: use pytest-timeout to avoid win...'. Below the title bar, it indicates '43 contributors'. The main content area shows a file with 89 lines (88 sloc) and 1.24 KB. The code is a requirements-dev.txt file. Lines 4-6 are grouped by a dashed box and labeled 'Dependencies'. Lines 7-15 are grouped by another dashed box and labeled 'Optional Dependencies'. The code includes comments at the top and a list of package versions with operators like >=, ==, and >.

Figure 2.1: The package usage scenario to develop pandas package and their different dependency types. According to the figure, pandas package has three main dependencies (i.e., numpy, python-dateutil, and pytz), and the rest dependencies are optional.

## 1.1 Usefulness of third-party packages

There are several benefits of third-party package usage in software development<sup>1</sup>. These are-

- **Save effort and time.** One of the main advantage of using third-party packages is that, it saves developers time and effort during software software development. Developers do not need to implement functions from scratch, instead they can focus on main business logic of the developing applications.
- **Use pre-tested code.** In term of testing, a big advantage of third-party package is the use of pre-tested code. Generally, popular packages are used

<sup>1</sup><https://www.scalablepath.com/blog/third-party-libraries/>

extensively in developing software by many developers. In this way, the packages receive many feedback on bug, and feature improvement requests. Therefore, using such pre-tested third-party packages guarantees the quality and stability of an application.

- **Use modular code.** Another advantage of third-party package usage is that, it facilitates developers to work with modular code. The package code is isolated from the rest of the application under development and communicates via a well-defined API to ensure the required functionality.

## 1.2 Drawback of third-party packages

Along with huge benefits of third-party packages, there are also several drawbacks. These are-

- **Dependency.** One of the major drawbacks of third-party package usage is dependency. Utilizing a package in developing application means that the code is coupled with that packages and its dependencies. Later, if a developer want to change the packages, he might have to perform significant changes to adopt new packages.
- **Lack of support.** Another potential risk is that the developer may not be able to maintain the package properly. A package need continuous maintenance to improve its features, soling bugs, and ensure compatibility. If the packages are not properly maintained, it will eventually face compatibility issues and may not perform optimally in new applications.,
- **Overuse.** The overuse of third-party packages in software development may cause dependency conflict and other related incompatibility issues which are difficult and time consuming to fix. In addition, overuse of packages will eventually result in increased memory space consumption and lower performance.
- **Security issues.** Another potential risk of third-party package usage is the possibility of affecting an application with security vulnerabilities.



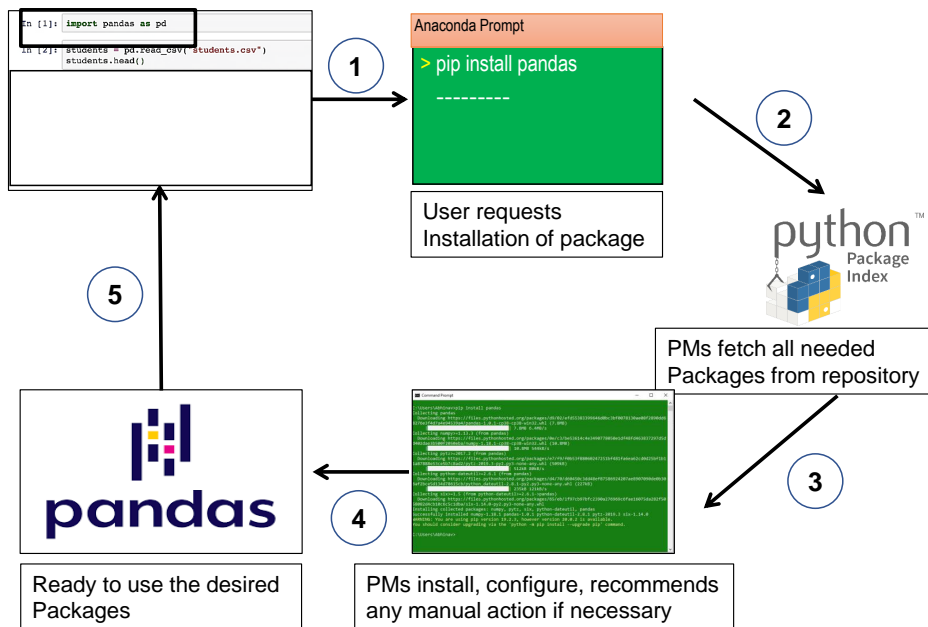


Figure 2.2: Typical workflow of third-party package management. It illustrates the workflow of python package manager pip to install the pandas packages from PyPI package repository.

## 2 Third-party Package Management

Package management is crucial to most technology stack in software development specially when building mobile and web applications. The package management system (i.e., package manager) is a tool that automates the process of installing, upgrading, configuring and removing of third-party packages from an application in a consistent manner [1, 76, 82]. The workflow begins with the user requesting the desired package using the package manager available on the system. Then the Package manager finds the requested package and its dependencies from the dedicated package ecosystem and downloads it. Finally, the package manager installs the desired package and advises on any manual actions that it finds necessary for successful installation [76]. Figure 2.2 shows the typical workflow of python package manager pip to install the pandas packages from PyPI package repository.

## 2.1 Basic Functions of Package Manager

The basic functions of third-party package manager are as follows:

- Act as a broker of package broker to extract desired package archives. The package managers serve over 5 million open source package to developers.
- Ensure package integrity and authenticity through Validating digital certificates and check-sums.
- Find, download, install, or update existing software packages from the authenticated package repository or app store.
- Group packages by feature/function to reduce developers confusion.
- Manage dependencies to ensure that packages are installed with all required packages and avoid dependency hell.

## 3 Software Packaging Ecosystem

The software packaging ecosystem is a collection of software components developed, shared, and evolved on a common technology platform, delivering numerous software services or solutions to meet consumer demand [25, 51, 57]. Due to the popularity of third-party package usage in modern software development, package ecosystems tend to be growing large that contain millions of packages [28]. For instance, npm (JavaScript language), PyPI (python language), maven (Java language) are popularly know package ecosystem among software developers.

### 3.1 Evolution of Software Packaging Ecosystems

Almost every programming language has come-up with a package ecosystem. Diversity of technology stacks and programming languages has led to variety of package ecosystems. For instance, npm brokers packages that run in the Node.js environment and are written in JavaScript, while the PyPI package ecosystem is built specifically to handle Python package dependencies. Figure 2.3 shows the overview of the evolution of thirteen third-party package ecosystems of top twenty

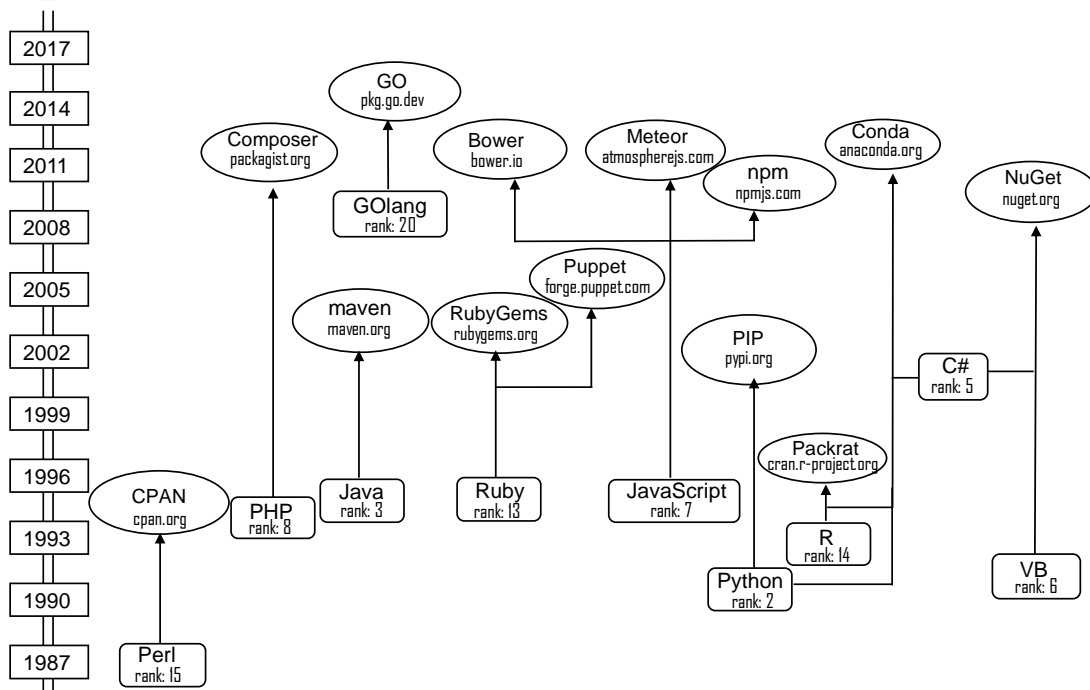


Figure 2.3: Overview of the evolution of thirteen third-party package ecosystems of top programming languages according to TIOBE index <https://www.tiobe.com/tiobe-index/>

programming languages. In the following, I will introduce different popularly known package ecosystems.

- **Maven.** ([maven.org](http://maven.org)) is a popular package ecosystem that serve third-party packages written in Java programming language to developers since 2004. The maven package ecosystem was created by Jason van Zyl and started as a sub-project of Apache Turbine in 2002. In 2003, it was accepted as a Apache Software Foundation project. Then, in July 2004, Maven’s was first released its v1.0. Later, Maven declared release of v2.0 in October 2005. In October 2010, Maven v3.0 which was backwards compatible with Maven v2. Currently, this packages ecosystem supports over 417,669 packages for developers.
- **CPAN.** ([metacpan.org](http://metacpan.org)) The comprehensive perl archive network is considered as the oldest package ecosystem. This CPAN package ecosystem

was developed in 1993 and has been active online to serve developers since October 1995. Currently, CPAN support over 38,459 packages to serve perl programming language based developers.

- **CRAN.** ([cran.r-project.org](http://cran.r-project.org)) The comprehensive R archive network is considered as the second oldest package ecosystem. In 1997, CRAN was created by Kurt Hornik and Friedrich Leisch. The CRAN archive contains all versions of the R distributions, its documentation, and the contributed R packages. For different operating system support like Windows and macOS, it includes both the source packages and pre-compiled binary files. CRAN support over 20,324 packages for serving statistical computation in R environment.
- **npm.** ([npmjs.com](http://npmjs.com)) The Node Package Management (that is, npm) is the fastest growing package ecosystem, delivering 186,6208 JavaScript packages to 11 million developers. The npm is committed to making JavaScript development elegant, productive and secure <sup>2</sup>. It consists of three different components: the website, the command line interface (CLI), and the registry. In 2010, the npm started its official package registry for JavaScript developers.
- **Packagist.** ([packagist.org](http://packagist.org)) is the main package repository for composer. The packagist ecosystem aggregates public packages which are written in PHP language and are install-able with Composer. In 2012, the packagist started its activity online to serve PHP developers. Currently, it serve over 316,855 packages to developers.
- **NuGet.** ([nuget.org](http://nuget.org)) is the official package repository developed by Microsoft organization for the .NET developer. NuGet package ecosystem is also popularly known as NuPack. Nuget starts its activity October, 2010. Currently, it supports over 264,221 packages for the .NET development platforms.
- **RubyGems.** ([rubygems.org](http://rubygems.org)) is the largest collection of packages written in Ruby programming language. The RubyGems development was started

---

<sup>2</sup><https://www.npmjs.com/>

in 2003 and was released to the public in early 2004. It support over 173,603 packages to serve Ruby developers.

- **PyPI.** ([pypi.org](http://pypi.org)) The python package index is the official third-party package repository for python. In 2003, the PyPI package ecosystem started its activity online to support python developers. According to [libraries.io](http://libraries.io), it support over 372,334 python packages.
- **Go.** ([pkg.go.dev](http://pkg.go.dev)) is the package ecosystem for golang programming language introduced by google. Golang is efficient, clean and expressive. The Go code modules facilitate concurrency mechanisms for multicore and networked machines, while its novel type system enables flexible and modular code construction <sup>3</sup>. Although golang was first introduced in 2009, the package management system of Go was one of our major focuses in 2018 <sup>4</sup>. According to [libraries.io](http://libraries.io), the go package ecosystem contain 390,438 packages to support developers.
- **meteor.** ([atmospherejs.com](http://atmospherejs.com)) is the package ecosystem dedicated for open-source isomorphic JavaScript web framework. It facilitates full-stack technology JavaScript language for developing complete web and mobile applications. Moreover, meteor includes a set of key technologies to build connected-client reactive applications, build tool, and a selected set of packages from the npm package ecosystem community <sup>5</sup>. In December 2011, the meteor was initially released under the name Skybreak. Finally, in April 2012, this framework was renamed as Meteor and officially active online. According to [libraries.io](http://libraries.io), the meteor package ecosystem currently contain 13410 packages to support full-stack JavaScript developers.
- **Conda.** ([anaconda.org](http://anaconda.org)) is a package ecosystem of python and R programming language to perform scientific computing. In july 2012, conda was developed by Anaconda, Inc..According to [libraries.io](http://libraries.io), this package ecosystem contain 12,763 packages to serve scientific computing.

---

<sup>3</sup><https://golang.org/doc/>

<sup>4</sup><https://go.dev/blog/modules2019>

<sup>5</sup><https://docs.meteor.com/>

## 4 Developers Discussion on Question-and Answering Site

Community based Question-and-Answer (Q&A) sites has become a new popular venue of searching and sharing knowledge. In the Q&A site, a user can ask/answer questions, and also provide feedback by commenting or voting to these questions and answers. Millions of users around the world utilize the Q&A sites to seek for the solution of their information needs [42, 49, 67, 102]. For example, Yahoo! Answers, Quora, Stack overflow, Stack Exchange are popularly known Q&A sites. Below, I introduce Stack Overflow, one of the most popular Q&A sites among developers.

### 4.1 Stack Overflow: A Popular Q&A Site Among Developers

Stack Overflow is a popularly known Q&A site among developers around the world to share programming related knowledge through asking a question, answering questions and participating in the discussions through commenting. Such Q&A process on the Stack Overflow platform eventually creates a crowd-sourced knowledge that helps developers around the world to build and improve their knowledge on programming and its related technologies. Previous studies by Gupta and Reddy [36], Storey et al. [83] reported that querying information from Stack Overflow is a regular activity by many developers. The Stack Overflow website consists of three main parts (i.e., question, answer, and comments). The success of this popular Q&A platform is evident through the fact that 92% of questions asked by users are getting answered, with median answering time of 11 minutes [55]. Parnin and Treude [70] reported that 84.4% of web searches for issues related to the jQuery API resulted in at least one Stack Overflow post on the first page indicating that programmers can access Stack Overflow for questions that have already been asked.

The image shows a Stack Overflow question page with several annotations. At the top, the title "How do I tell CPAN to install all dependencies?" is enclosed in a dashed box with an arrow pointing to a label "Title". Below the title, the question text is shown, with a score of 120 on the left side, also enclosed in a dashed box with an arrow pointing to a label "Score". The question body contains a code block for CPAN configuration and a paragraph of text. Below the question, there are tags for "perl", "dependencies", and "cpan", with an arrow pointing to a label "Tag". At the bottom, the asker's information is shown, including the name "Nifle" and the date "asked May 22 '09 at 16:36", with an arrow pointing to a label "Asker". The body of the question is also annotated with an arrow pointing to a label "Body".

Figure 2.4: Example of Stack Overflow question (Id: 898782) where a developer ask for the solution of CPAN dependency management.

## Question

The Stack Overflow question-and-answer process starts with a user posting a question that related to programming or other related issues. A question is posted in Stack overflow to describe problem faced by a user. A question body may contain different code snippets, log files, command line scripts, software screenshot etc. The main goal of posting a question is to request for information needs and solution from the community. Figure 2.4 shows an example of question asked by a user on a issue related to installation of CPAN dependencies.

## Answer

An answer of a question is the solution or a set of instruction that help the users to solve specific problems. The answer provided by other users can be

34

Here is the one-liner making these changes permanent including automatic first-time CPAN configuration:

```
perl -MCPAN -e 'my $c = "CPAN::HandleConfig"; $c->load(doit => 1, auto
```

Or combine it with `local::lib` module for non-privileged users:

```
perl -MCPAN -Mlocal::lib=~/perl5 -e 'my $c = "CPAN::HandleConfig"; $c->
```

Run it before using the CPAN shell or whatever.

Share Edit edited Jan 22 '17 at 11:19 answered Feb 12 '14 at 18:18  
 Follow Peter Mortensen 28.9k ● 21 ● 96 ● 123 Atento 746 ● 6 ● 7

isn't it the same as in @sdf anwer? – filimonov Feb 23 '16 at 13:46 Comment

---

3

I found this to be, by far, the quickest and most reliable way to install CPAN modules:

```
yes | perl -MCPAN -e "CPAN::Shell->notest(qw!install Your::Module!)"
```

Share Edit answered Jan 21 '17 at 11:19 answered Jul 29 '15 at 14:08  
 Follow Peter Mortensen 28.9k ● 21 ● 96 ● 123 Vladimir Marchenko 31 ● 2

Answerer

Figure 2.5: Example of Stack Overflow answers of a question (Id: 898782) where other Stack Overflow users propose solution for the CPAN dependency management.

example source code, command line scripts, external tutorial or document links, plain text, step by step instructions, etc. Sometimes, one question can receive multiple answer proposal from different users. If a proposed answer can satisfy requirements to solve the specific problem, then the user who ask the question may mark the answer as accepted answer. Once a question receive its accepted answer, other users of Stack Overflow can still contribute to the question by proposing alternative answers or by editing other existing answers. Figure 2.5 shows the answers of a question, where other Stack Overflow users propose solution for the CPAN dependency management.



## Comments

The comments are feedback and discussion by Stack Overflow users within questions and answers. In question, comments are used to ask for further information, suggest similar questions, and suggest improvement of questions, etc. Again, in answers, users participate in more explanation of answers, or better alternative solutions.

## 5 Definition of Developers Experience

In contemporary software development, the usage of third-party has become a common practice due to its huge benefits. However, managing third-party packages is important for most technology stacks in software development. Developers often face various issues (like installation, configuration, testing, etc) when they use third-party package in software development. In this thesis, the term *developers experience* means the type of challenges faced and information needs by developers during third-party package usage from different software ecosystems. For instance, Figure 2.4 shows that a developer experiences problem related to installing CPAN package dependencies.

## 6 Related Works

Complementary related works are presented throughout the thesis. This section describes some additional related works.

**Dependency management studies.** Prior studies on third-party package management showed that developers struggled to manage their software dependency [3, 17, 18, 26, 28, 35, 41, 52, 72, 87]. In detail, Bogart et al. [17, 18], performed multiple case study on a set of software ecosystems with different tooling policies. They found that developer practices differ significantly between ecosystems. All ecosystems share values such as stability and compatibility, but others are different. In addition, all communities are interested to invest in tooling to facilitate dependencies maintenance. Kikas et al. [41], analyzed dependency network of three software ecosystems (i.e., JavaScript, Ruby, and Rust). They

reported that there exist significant difference in dependency network structure across language ecosystems. Decan et al. [26, 28] studied several software ecosystem and report that dependency network tend to grow over time in term of size and number of packages. In addition, they observed that a minority of packages are mostly responsible for the package updates. Abate et al. [3] studied state-of-art package managers to investigate dependency solving capacity. They reported that solving dependency problem is challenging. German et al. [35] and Lungu et al. [52] investigated issues related to dependency graphs and dependency management specifications. They reported that dependencies also exist between projects in a software ecosystem. Raemaekers et al. [72] and Teyton et al. [87], have shown that dependency management involves making cost-benefit decisions related to keeping package dependencies up to date.

Some studies reported that usage of outdated dependency in software can have security vulnerability and backward incompatibility issues [24, 29, 40, 71]. In detail, Cox et al. [24] reported increased security and backward compatibility risks when relying on older packages. They found that systems using old dependencies were four times more likely to have security issues than systems with updated dependencies. Jafari et al. [40] examine the npm ecosystem for evidence of recurring dependency management issues that cause many problems, including security threats, bugs, dependency breakage, runtime errors, and other maintenance issues. Derr et al. [29] conducted a survey on 200 application developers to investigate the impact of outdated package usage. They reported that most apps are using older packages, and nearly 98% are actively using package versions that are affected by known security vulnerabilities. Pashchenko et al. [71] conducted a survey on software developers to shows the highlight developers effort to mitigate vulnerabilities. They reported that bundling security fixes with functional changes could hinder adoption due to lack of resources to fix critical changes.

Moreover, other studies [15, 73, 79] show dependency updates at the API level is slow and lagging. Robbes et al. [73] studied deprecation's of API that led to ripple effects through the entire ecosystem. They report that deprecation of APIs can have a significant impact on the entire ecosystem in terms of broken dependency. Bavota et al. [15], studied 147 Java Apache projects to understand the changes that trigger dependency upgrades (bug fixes trigger upgrades, but

changing interfaces makes them less likely to upgrade). They report that when a new release of a project is published, an upgrade will be triggered if the new release contains major changes (such as new features / services) and a large number of bug fixes. Instead, developers hesitate to perform an upgrade when some APIs are removed. Sawant et al. [79], conducted a study on API deprecation mechanism by considering 25,000 clients of five popular Java APIs on GitHub. They highlighted the surprising similarities between how API clients are updated and how Smalltalk and Java react.

Other studies mined and identified patterns in the migration of third-party packages [27, 30, 43, 44, 45, 63], not only between client applications and libraries, but their effect at the ecosystem level. Kula et al. [45], mined developer responsiveness to existing security awareness mechanisms on 850K library dependency migrations from 4,659 GitHub projects. They found that developers were particularly reluctant to update third-party libraries to fix vulnerabilities. Decan et al. [27], mined 400 security reports over 6 years on an npm dependency network containing over 610k JavaScript packages. They report findings and provide guidelines for package maintainers and tool developers to improve the process of addressing security issues. Mirhosseini and Parnin [63], mined 7,470 GitHub projects that use notification mechanisms to identify changes in upgrade behavior. On average, projects with pull request notifications were upgraded 1.6 times more often than non-tool projects, according to their results. Dietrich et al. [30] studied seventeen different package ecosystems. Their findings reveal that developers struggle to find a sweet spot between fixed version dependency predictability and flexible dependency agility.

While these studies have shown that developers struggle to migrate their dependent packages, the common assumption is that Package management systems broker package dependencies without any issues. This thesis investigates the issues confronted when using the third-party package from different software packaging ecosystems.

**Software build and configuration studies.** Build systems are indispensable part of software evolution, which has gained attention in recent times. There have been several studies on software build systems [6, 22, 53, 59, 60, 61]. McIntosh et al. [61], studied low-level abstraction-based framework-driven build tech-

nology and tools to automatically manage external dependencies. According to their research, the latest framework-driven build technologies need to be maintained more often, and these build changes are tightly coupled with the source code than low-level or abstract-based ones. Again Adams et al. [6] found that support for both reverse and re-engineering techniques are needed to maintain build system evolution. McIntosh et al. [60] analyzed the overhead that build maintenance imposes on developers, finding that build maintenance is costly. McIntosh et al. [59] also evolution of java build systems based on static and dynamic perspective. They studied build system specification complexity using software metrics. They also examined the complexity and coverage of representative build runs. Again, Macho et al. [53] studied the the detailed reasons for the dependency-related build breakage and then propose automatically repair the build. Chernikova and Shalaev [22] proposed an approach that combines a directed acyclic graph model of package dependencies management and task queue system. The experimental results have confirmed that the use of the proposed distributed build system reduces build time significantly.

**Stack Overflow Studies.** Q&A platform like Stack Overflow has gained huge research interest, with topics relating to community dynamics, technical issues of programmers and human factors [62]. Several empirical case studies were performed using Stack Overflow data such as improving API documentation and usage scenarios [68, 93, 95], new programming language (Go, Rust, and Swift) related discussion [19], privacy [84], assess technical debt in software project [33], Docker development [38], IOT [94], code weakness [105], programming errors [20], mobile platforms [99], code reuse [100], deep learning framework [37], etc. Some studies were done on human factors like IT skill [64], programmers expertise [31], etc.

Several tool supports and recommendation models were developed using Stack Overflow data resources such as PostFinder [78], bug severity prediction model [85], code2Que [34], sc++ [10], SONAS [98], CAPS [8], IEA [97], idev [32], Icsd [103], etc. These studies reported that Stack Overflow data resources are useful to solve developers challenges.

In this thesis, the same data source (i.e, Stack Overflow) is used but different from the above mentioned empirical studies. To the best of my knowledge, there

is no prior work that conducted study on third-party package discussions from Stack Overflow. I extracted third-party package Q&A posts from Stack Overflow and perform a series of empirical studies. I believe that this studies highlight the challenges and information needs associated with using third-party packages from software packaging ecosystem for developers, package manager designers and researchers.

# 3 | Package Management

## 1 Introduction

The purpose of this Chapter is to investigate developers experience (i.e., challenges) on third-party package management in different software packaging ecosystems. To accomplish the goal, this chapter departs with two consecutive studies. At first, I performed a quantitative study on 497,249 Stack Overflow Q&A posts from 13 packaging ecosystems to contrast developers experience of package management. Second, I conducted a qualitative study on 1131 third-party package question posts from 3 packaging ecosystems to get deeper understanding of the questions asking and their underlying causes.

Among the main findings, the quantitative study identifies 10 topics that developers are discussing on third-party packages which are further categorized under three major themes: (i) Package management, (ii) Input/Output, and (iii) Package. Contrasting developers response in terms of topic popularity and difficulty, I find that third-party package posts that relate to error and testing topics are most popular and also most difficult for developers to solve. In addition, contrasting developers experience based on package management features, I observe that developers experience is different, depending on the choice of packaging ecosystem. After combining the results of quantitative study, I speculate that developers using Go and Meteor package ecosystems have relatively easier time finding answers to their questions on Stack Overflow compared to developers using other package ecosystems.

Again, the qualitative study identifies that third-party package related questions arise due to lack of instructions (i.e., around 31 to 42% of question samples)

and error messages (i.e., around 27% to 37% of questions). Further investigation on third-party package posts shows that the underlying causes can be categorized into package management tool usage, general dependency practices, specific migration, and others. An analysis of the underlying causes in 3 packaging ecosystems reveals that most questions related to third-party packages by developers are due to lack of knowledge on package management tool usage, rather than specific migration.

The rest of this chapter is organized as follows. Section 2, presents quantitative study to contrast developers experience in package management. Section 3 presents the qualitative study to investigate the root causes of package management discussion and their kinds of questions. Section 4 discusses the implications of this chapter. Section 5 presents the possible threats to validity of results. Finally, in Section 6, I summarize the results.

## 2 Contrasting Developers Experience in Package Management

In this section, I quantitatively examined developers experience on third-party package management in different packaging ecosystems. To achieve this goal, I analyzed the third-party package related discussion on Stack Overflow since it provides a rich dataset and also been used by similar studies in other domains, such as Chatbot [5], Docker [38], Mobile [77], etc. This quantitative study was accomplished by (I) Selecting package ecosystems and their features, (II) Building third-party package related discussion dataset, (III) Question Topic Modeling, and (IV) Contrasting developer discussion topics relating to different ecosystems from the two perspectives. Figure 3.1 illustrates the overview of contrasting developers experience in third-party package management on 13 selected package ecosystems from top 20 programming languages. In the following, I explained each of them in detail.

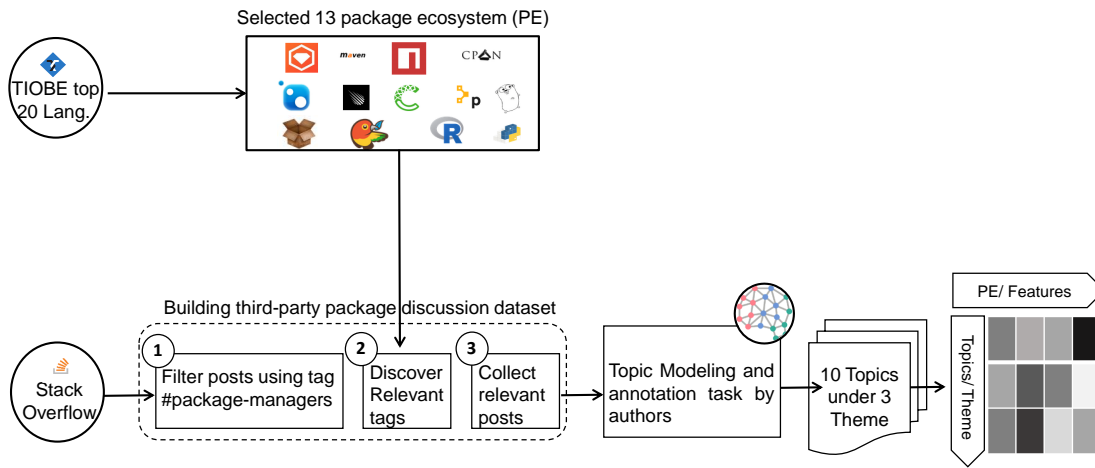


Figure 3.1: An overview of contrasting developers experience in third-party package management on 13 selected package ecosystems from top 20 programming languages.

## 2.1 Selecting package ecosystems and their features

To explore the most popular third-party package ecosystems, I started from the top twenty programming languages from tiobe.com<sup>1</sup> as of June, 2021. To extract the package ecosystems for each language, I used `libraries.io`<sup>2</sup> to gather the package ecosystems that are related to these programming languages. Note that some of the package ecosystems do not have a dedicated package manager, and that there can be several package ecosystems that are written in one programming language. Other features include the dependency tree<sup>3</sup> and environment.

Table 3.1 shows a summary of features that are specific to each of our package ecosystems. In detail, I find that nine of our package ecosystems support a specific programming language (i.e., Maven for Java, npm and Bower for JavaScript, Go for GoLang, RubyGems for Ruby, PyPI for Python, CPAN for perl, CRAN for R, Packagist for PHP). The other package ecosystems support multiple programming languages. As shown, a package ecosystem serves from 6,900 to over 1.8 million

<sup>1</sup>Details of the dataset are available at <https://www.tiobe.com/tiobe-index/>

<sup>2</sup><https://libraries.io/>

<sup>3</sup>An example of differences in dependency trees is described at <https://npm.github.io/how-npm-works-docs/npm3/how-npm3-works.html>



Table 3.1: Summary of each package ecosystem and their features.

Package Ecosystem	Programming Language	Tiobe Rank	Environment	Dependency Tree	Package Archive link	# of packages in ecosystem
PyPI	Python	2	Python	Flat	pypi.org	372,334
Maven	Java	3	JVM	Flat	Maven.org	417,669
Bower	JavaScript	7	Node.js	Flat	bower.io	69,625
Meteor	JavaScript	7	Node.js	Nested	atmospherejs.com	13,410
npm	JavaScript	7	Node.js	Nested (v2)	npmjs.com	1,866,208
Packagist	PHP	8	PHP	Flat	packagist.org	316,855
Puppet	Ruby	13	Ruby MRI	Flat	forge.puppet.com	6,923
RubyGems	Ruby	13	Ruby MRI	Flat	rubygems.org	173,603
CRAN	R	14	RStudio	Flat	cran.r-project.org	20,324
CPAN	Perl	15	Perl	Flat	metacpan.org	38,459
GO	Golang	20	Go	Flat	pkg.go.dev	390,438
NuGet	C#, VB	5, 6	.NET	Flat	nuget.org	264,221
Anaconda	Python, R, C#	2, 14, 5	Anaconda	Flat	anaconda.org	12,763

packages.

## 2.2 Building third-party package related question-and-answer dataset

First, I downloaded the Stack Overflow data dump published on SOTorrent [13]. The data dump contains all question-and-answer including post metadata such as creation date, view count, favourite count, score, etc. After parsing the Stack Overflow data dump, the initial dataset had 39.83% (18,699,426) question posts, and 60.17% (28,248,207) answer posts from July 2008 to December 2019. Since Stack Overflow does not provide any fine grained dataset, I needed a way to identify third-party package related posts for the target ecosystems. To accomplish this goal, I utilized tag-based question post filtering which was also used by prior studies [5, 77]. In detail, the third-party package related question-and-answer dataset building was performed through three distinct steps:

- Step 1: Filter using #package-managers tag. In Step 1, my intention was to filter posts that are only related to selected 13 package ecosystems. To accomplish this, I started with collecting question posts that were tagged

with the keyword `#package-managers`, which is described in Stack Overflow, as software that allows administrators (and in some cases also users) to control the installation and upgrade process of packages on their systems. The output of Step 1 was 806 question posts.

- Step 2: Discover relevant tags. In Step 2, I extracted a list of further relevant tags from the 806 question posts (Step 1). A potential risk of expanding the list of tags was the possibility of introducing noise. For example, “*NPM search remote packages*” is a relevant post that contains broad tags such as `javascript` and `node.js`. I used a semi-automatic method to mitigate this issue and manually removed tags that are too broad. I along with another authors manually checked 626 unique tags. We observed that some tags—although having low frequency (i.e., occur only one time) in the initial posts dataset—were highly related to our 13 package ecosystems as shown in Table 3.1. For instance, `maven-2` is a tag that occurs only one time as a co-existing tag in the initial package ecosystems post dataset, but the tag itself is associated with many more posts (5,568 question posts). The output of Step 2 was a list of 28 relevant tags for the selected 13 package ecosystems as shown in Table 3.2.
- Step 3: Collect relevant posts. In Step 3, I used the list of relevant tags from Step 2 to collect final posts. The output of Step 3 was a cleaned third-party package related question-and-answer dataset with 497,249 posts, where 214,609 were questions and 282,640 were answer posts.

## 2.3 Question Topic Modeling

To explore topics from third-party package posts dataset, I applied the popular Latent Dirichlet allocation (LDA) topic modeling technique on posts title. This process was accomplished through three distinct steps.

- Step 1: Extract post title and preprocess. In step 1, I applied a filtering technique to remove irrelevant information. In topic modeling, I focused only on the title of the question post, as the body of the post could add

Table 3.2: List of the 28 manually validated tags discovered from Step 2 of the data preparation.

Initial tag	Identified relevant tags
package-managers	npm, nuget, bower, conda, nuget-package, anaconda, go, maven, rubygems, bower-install, npm-install, packagist, pypi, miniconda, npm-scripts, cran, meteor, npmignore, pnpm, npm-shrinkwrap, godeps, go-modules, meteorite, nuget-package-restore, cpan, puppet, maven-2

noise to the analysis. A similar approach to the previous study [77] was used to extract the title of the post and preprocess the data. This includes removing email, newline characters, and stop words using regular expression<sup>4</sup> and python NLTK.<sup>5</sup> Then I used Gensim to create a bigram model<sup>6</sup> and lemmatize the words to map the original words. The output of this step was the third-party package related question post title corpus used as input for the LDA model.

- Step 2: Identify third-party package discussion topic using LDA modeling. In step 2, I used the post-title corpus obtained in step 1 to identify third-party package discussion topics. To obtain the topic names, I utilized the LDA model [16], which was also used by previous studies [23, 50, 77, 104]. In this study, the Mallet model of LDA was adopted [58] to create group of posts based on the keywords exists in the post-title corpus. To obtain the optimal topics number  $k$ , I repeatedly run the model through several iterations. In detail, first, I run the LDA model for a specific range (0-50) with 3 step size increment. Second, I selected the next sub-optimal range (12-25) based on the coherence score [77]. Third, I again run the same model for the selected range (12-25) with 1 step size increment and thus come-up with 15 topics. Finally, I run the model with topic number  $k = 15$

<sup>4</sup>Regular expression: <https://docs.python.org/3/library/re.html>

<sup>5</sup>Python NLTK: <https://www.nltk.org/>

<sup>6</sup>Gensim model: <https://radimrehurek.com/gensim/>

to get 15 third-party package discussion topics with related keywords (10 keywords per topic).

- Step 3: Label and merge third-party package discussion topics. In Step 3, I sorted posts based on topic prediction contribution. Then I along with two independent authors manually inspected the top 10 keywords from each topic by following instruction from Agrawal et al. [7]. This includes reading most relevant 15 posts (based on higher topic prediction score) for each topic to identify a topic name that best explain keywords and question posts of the topic. In addition, we manually inspected 10 randomly selected posts from each topic to see if the selected topic name fits the posts properly. After completing the naming process, we observed some topic can be merged since they were very closely related. Therefore, through a round-table discussion, we merged similar topics into one topic name that were closely related. Thus, we obtained 10 third-party package discussion topics from 15 suggested topics by LDA topic modeling process. Finally, we grouped the third-party package discussion topic into 3 major themes. The output of Step 3 was 10 unique third-party package discussion topics mapped under 3 major themes. Table 3.3 and Figure 3.2 illustrate the 10 third-party package discussion topics, major themes and their associated keywords. From the Figure 3.2, it is clear that most software developers ask questions about package management. In detail, developers ask questions on dependency, build, configuration, error, testing where all of these topics have a higher percentage value than the average value for topics (10.00%). All of these 10 third-party package discussion topics are grouped together under 3 main themes (i.e., Package Management, I/O, and Package). In the following, I discussed each themes along with their topics, sample question posts from Stack overflow for clear explanation.

- **Package management.** The package management process consists of installing, configuring, updating, removing a verified third-party package from an application in a consistent manner. It is the largest theme (62.61%) among the three themes related to third-party packages discussed by developers on stack overflow. The results indicate

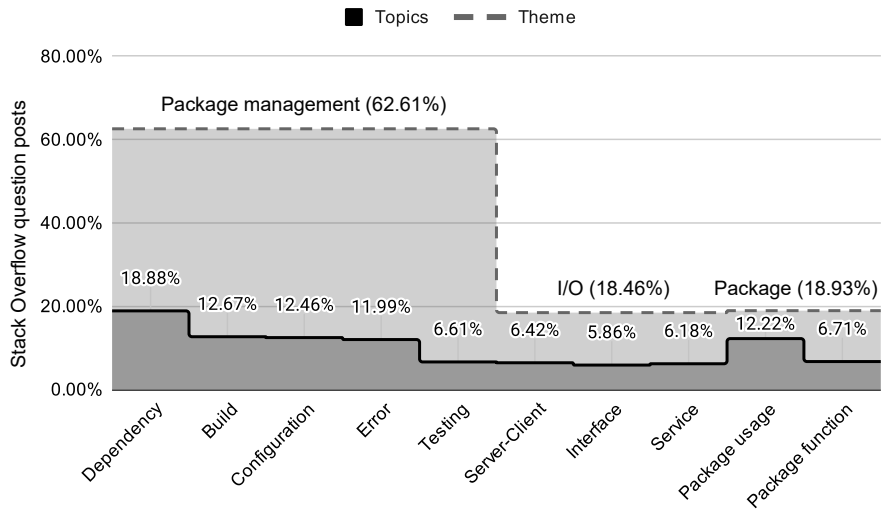


Figure 3.2: Distribution of third-party package related discussion topics and their higher level theme (in percentage).

that developers are struggling with package management while using packages from different software packaging ecosystems. This concern is more strengthened by [3, 17, 18, 72, 87]. These studies investigated software packaging ecosystem policies, cost-benefit tradeoffs in package management and reported that solving package management issues are hard for developers. The topics obtained under this theme are, dependency, configuration, software build, testing, and errors. In the following, I have explained each topic under this theme.

- \* *Dependency.* In this topic developers ask questions related to dependency migration like package version upgrade, downgrade, replacing existing package with another new package, and other related issues. Under package management theme, dependency is the most talked topic by developers. This finding is consistent with previous studies [30, 45] that showed that developers struggle to smoothly migrate their package dependencies. For instance, this post (id:10507335, title: Maven requires manual dependency update?) is related to dependency update.
- \* *Build.* In this topic developers ask question related to software build process like jar file generation, make file, compiling projects,

and related failure scenarios. Under package management theme, build is the second largest topic discussed by developers. Successful build is indispensable part of package management. Prior studies also concern with build problems [48, 80], build maintenance efforts [53, 60]. For instance, this post (id:264120, title: How to move build task into parent POM) is related build task.

\* *Configuration.* In this topic developers ask questions related package configuration, setting up installation environment, and other related issues. Under package management theme, configuration is the third largest topic discussed by developers. Several studies discussed about complexity of install/uninstall problem [11, 92], system configuration smell [81], etc. For instance, this post (id:103918, title: Automate a Ruby Gem install that has input) is related to configuration topic.

\* *Error.* In this topic developers ask questions related to various errors and exceptions raised during third-party package based software development. Under package management theme, error is the fourth largest topic discussed by developers. Several studies reported that while managing third-party package developers often face errors and backward incompatibility issues [24, 40]. For instance, this post (id: 577377 , title: “Missing artifact” errors with company repository) is related to error topic.

\* *Testing.* In this topic developers ask question related to testing process of software projects, integration, and other related failure scenarios. Under the package management theme, testing is the least asked topics by developers. Previous studies [65, 86] shows the importance of testing in survival analysis of package and identifying the breaking updates. For instance, this post (id:2272239, title: Testing running over an half hour) is related to unit testing of maven-2.

– **Package.** The package consists of code modules, configuration files, and information about its dependencies. It is the second largest (18.93%) theme in our third-party package related posts. Previous

studies [14, 75] discussed the increasing popularity of third-party packages in modern software development. The topics obtained under this theme are, package usage and package functions. In the following, I explained each topics under this theme.

- \* *Package usage.* In this topic developers ask question about the usage scenarios of third-party packages, tutorials, usage documentation, etc. Under the package theme, the package usage is the largest topic discussed by developers. This finding is further verified by this study [46] discussed that the importance of package usage information, including tutorials, usage documentation, and real life code examples, along with Q&A websites like Stack Overflow for developers. For instance, this post (id:4461344, title: Looking for the documentation of the maven-glassfish-plugin) is asking for reference documentation.

- \* *Package function.* In this topic developers ask question about the basic functionalities of a third-party package and implementation of a new module or method, passing arguments to a function, and other related issues. Under this theme, package function is the least asked topics by developers. For instance, this post (id:5926003, title: How do I hold onto a factory function in Go?) is related to package function.

- **Input/Output.** The Input/output refers to services that helps users to all input output operations. It is the smallest (18.46%) theme in our third-party package related posts. The topics obtained under this theme are, application deployment services, client-server communication, and interfacing issues. In the following, I explained each topics under this theme.

- \* *Service.* In this topic developers ask question about different applications development and deployment related issues. Under the Input/Output theme, service is the second largest topic discussed by developers. For instance, this post (id:602511, title: Maven: How to deploy with deploy-file and custom wagon) is related to deploying a service of maven.

- \* *Server-client.* In this topic developers ask question related to client-server connectivity, user question and response, message transmission, user permission, etc. Under the Input/Output theme, server-client is the largest topic discussed by developers. For instance, this post (id:781527, title: Maven server authentication as profile properties) related to sever authentication.
- \* *Interface.* In this topic developers ask question related to interfacing like read-write operation, file processing, accessing channel, etc. Under the Input/Output theme, interface is the least asked topics by developers. For instance, this post (id:3398490, title:Checking if a channel has a ready-to-read value, using Go) is related to interfacing issue of Go package ecosystem.

**Takeaway 1:** Developers questions related to third-party package can be clustered into ten different topics including dependency, build, configuration, error, testing, service, server-client, interface, package usage and package functions, etc. These topics can be further grouped into three major themes as package management, input/output, and package.

## 2.4 Contrasting developer discussion topics relating to different ecosystems

Here, I contrasted the developer discussion topics associated with using packages in software ecosystems from two different perspectives. These are (a) Topic popularity and difficulty, and (b) package management features.

### A. Contrasts in Responses: Popularity and Difficult

**Approach** To characterize third-party package discussion topics in terms of their popularity and difficulty, I adopted metrics as defined by Yang et al. [101]. I used the post score, views count, favourite count to measure the popularity of topics. On the other hand, answer count, percentage of accepted answer count, comments count and PD score were used to measure the difficulty of a topic. These metrics



Table 3.3: Topics and their top ten keywords extracted from third-party package related question posts. The topics are categorized into three themes.

Theme	Topic Id	Topic Name	Sample Keywords
Package Management	0,1	Dependency	dependency, version, specific, release, artifact, resolve, late, update, change, repository
	6, 8	Build (2)	build, project, create, generate, make, jar, war, compile, failure, resource
	3, 11	Configuration (2)	package, add, reference, install, set, environment, variable, import, module, library
	7	Testing	run, test, command, fail, execute, integration, unit, report, clean, surefire
Input-Output	9, 14	Error (2)	error, throw, give, exception, load, unable, fix, issue, work, problem
	10	Services	application, deploy, web, spring, app, tomcat, deployment, service, engine, boot
	5	Server-client	server, client, user, request, http, connection, access, response, message, proxy
Package	2	Interface	file, time, read, write, image, output, channel, log, process, multiple
	13	Package function	type, function, struct, string, variable, interface, slice, method, argument, pass
	4, 12	Package usage (2)	template, react, collection, event, render, helper, database, component, field, document

are all well-known and also used in previous papers [5, 77]. I utilized median value of each metric to investigate popularity and difficulty of package ecosystem topics. In the following, I used the stack overflow tour <sup>7</sup> to define the topic popularity and difficulty metrics.

- The *score count* per post indicates the usefulness of a post to the community. According to Stack Overflow tour, members are allowed to up-vote posts that are considered useful to developers. This vote is summarized as a score. I used this score as one of the metrics to measure the usefulness of topics.
- The *views count* per post indicates community interest. If a post is highly views by both registered and unregistered users, then the post can be inferred as a popular post among developers. Therefore, this metric shows the interest/popularity of the topics.
- The *favourite count* per post indicates problem and solutions that developers found helpful. The average post score is interpreted as usefulness to the developer.
- The *answer count* per question post indicates the difficulty to answer a question by the other Stack Overflow users.
- The *accepted answer count* indicates whether an answer satisfy to solve users problem. Among the answers of a question post, the original author of the question can mark an accepted answer if the answer has its merit to solve his problem.
- The *PD score* indicates the difficulty to answer a question in a specific topic. To calculate the PD score, I follow the same procedure as defined by Yang et. al. [101]. First, I extract the answer count and the view count per questions of each topic. Then, I calculate the median answer count and view count to find the PD score as formulated below:

$$PD\ score = \frac{Answer\ count\ (median)}{View\ count\ (median)} \times 100\% \quad (3.1)$$

---

<sup>7</sup>stack overflow tour: <https://stackoverflow.com/tour>

Table 3.4: Popularity of third-party package discussion topics. All the metrics are represented in median value. The median value of favourite count for all topics are zero. Result shows that error and testing related issues are most popular among developers.

Theme	Topics	Score count	Views count
Package Management	Dependency	1	420
	Build	1	409
	Configuration	1	348
	Error	1	447
	Testing	1	440.5
Input-Output	Server-Client	1	319
	Interface	1	277
	Service	0	405
Package	Package usage	0	216
	Package function	1	284

In general, a low number of answers and a large number of views for unanswered questions indicates that only a few users of Stack Overflow can answer the questions. Therefore, I used PD scores to measure the difficulty of the questions answered in a particular third-party package discussion topic. The lower the PD score, the harder it is for other Stack Overflow users to answer the question.

**Results.** Table 3.4 and 3.5 show the popularity and difficulty of third-party package discussion topics. I observe that popular topics are also difficult to answer by other Stack Overflow users. According to the Table 3.4, I find that overall error and testing related issues are most popular among developers. Again, according to Table 3.5, I find that error topic related issues are most difficult to answer while package usage issues are easiest to answer by other stack overflow users. In detail, among the topics under package management theme, error (median views count 447) is the most popular topic followed by testing (median views count 440.50)

Table 3.5: Difficulty of third-party package discussion topics. The answer count metric is represented in median value and the accepted answer count and PD score are represented in percentage (%). Result shows that error topic questions are most difficult to answer while package usage issues are easiest to answer by other stack overflow users.

Theme	Topics	Answer Count	Accepted answer count (%)	PD Score (%)
Package Management	Dependency	1	48	0.24
	Build	1	46	0.24
	Configuration	1	45	0.29
	Error	1	44	0.22
	Testing	1	45	0.23
Input-Output	Server-Client	1	48	0.31
	Interface	1	53	0.36
	Service	1	43	0.25
Package	Package usage	1	53	0.46
	Package function	1	64	0.35

and dependency (median views count 420). In term of difficulty, error (PD score 0.22) topic related issues are most difficult to answer followed by testing (PD score 0.23) and build (PD score 0.24) in the package management theme. In the input/output theme, service is the popular (median views 405) and also difficult (PD score 0.25) to answer followed by server-client and interface topics. In the package theme, package function is the most popular (median views count 284) and also difficult (PD score 0.35) to answer followed by package usage.

**Takeaway 2:** Different topics imply different degrees of difficulty. Error and testing topic related issues are most popular among developers and also most difficult to answer by others. In addition, I find that, package usage topic related issues are least difficult to answer.

## B. Contrasts in Features: Topics and Features

**Approach.** To understand whether the package management features of package ecosystems have correlation with developers challenges/experiences, I utilized heatmap colored cells to show a two-dimensional matrix between the topics and the features of the package ecosystems. In detail, first I performed mapping of each package ecosystem related posts obtained in Step 3 of question modeling 2.3 based on the package ecosystem name, and their technology stack like programming language, environment, supported dependency tree, etc. For instance, a package ecosystem post tagged with npm is identified as a post related to npm package ecosystem, and its technology stack is (JavaScript, node.js, and nested dependency tree). In the same way, I characterized each Stack Overflow post by package ecosystem name and their technology stacks. Afterward, I used heatmap visualizations to compare the differences between the features of a package ecosystem and the kinds of topics that developers ask on Stack Overflow. In the heatmap, I reported the frequency counts of each dimension that is reflected in the colored cells.

**Results.** Figure 3.3, 3.4, 3.5, 3.6, 3.7 illustrate the results of package ecosystem and its features correlation with developers experience. I observe that developers from different package ecosystem report different issues. In addition, analysis results indicate that specific feature of a package ecosystem has correlation with developers experience. The analysis results are explained into two fold: (i) Contrasting package ecosystem and topics, and (ii) Contrasting package ecosystem features and topics.

- (i) *Contrasting package ecosystem and topics.* Figure 3.3 shows that the topics related to each package ecosystem differ. For example, package ecosystems like CRAN, CPAN, and Conda tend to attract configuration related questions, while Go and Meteor have their developers ask questions related to the package function and usage. Under the broader themes, the results are consistent with the results of Table 3.5 and Figure 3.4, as most topics are related to package management. This evidence suggests that developers using Go and Meteor package ecosystems may face different types of issues when compared to developers using other package ecosystems.

**Takeaway 3:** Developers from different package ecosystems report different issues. Findings indicate that RubyGem developers report errors, while NuGet developers report configuration issues. Combining with takeaway 2, developers using the Go and Meteor package ecosystems face relatively easy experience.

- *(ii) Contrasting package ecosystem features and topics.* Taking a deeper look at the features, I can see that developers using package ecosystems built for JavaScript technologies and environments tend to encounter different types of issues, whereas developers using package ecosystems built for the Python, R and Perl languages and related environments tend to focus on configuration issues (cf. Figure 3.5 and Figure 3.6). In terms of the dependency tree, Figure 3.7 shows that flat dependencies tend to attract dependency related issues, while package ecosystems with nested dependency trees encounter package usage related issues. One possible explanation is that nested dependencies are a solution to ‘dependency hell’, thus removing dependency related issues for package ecosystems with nested dependency trees<sup>8</sup>.

**Takeaway 4:** Developers from different programming languages report different kinds of issues. I find that applications that are developed using the Python language reported more configuration issues, when compared to Ruby developers who report errors.

---

<sup>8</sup>This issue is discussed by this npm blog post at <https://npm.github.io/how-npm-works-docs/theory-and-design/dependency-hell.html>

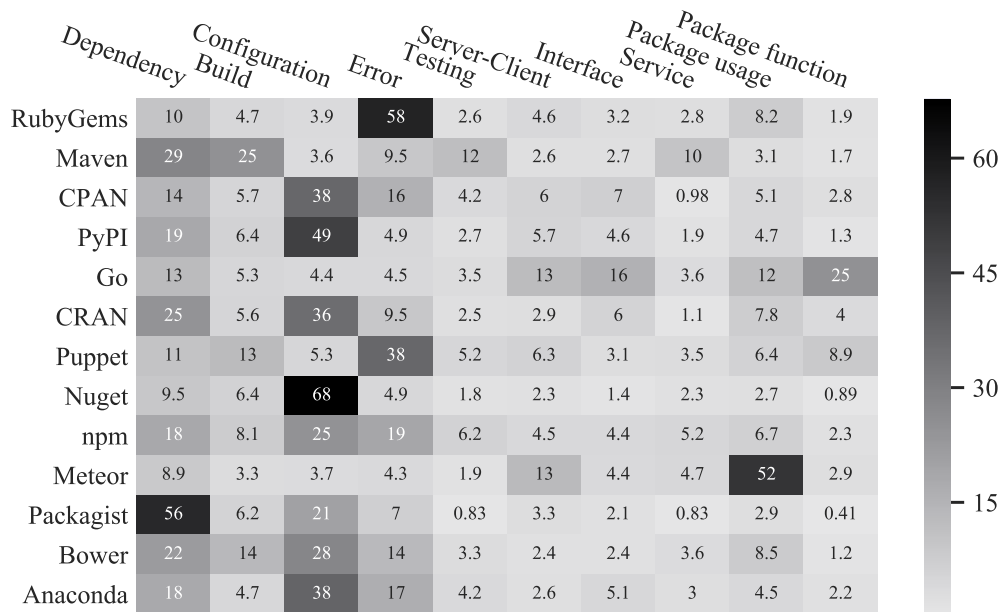


Figure 3.3: Package ecosystem vs. topic, where developers of different package ecosystems ask different questions.

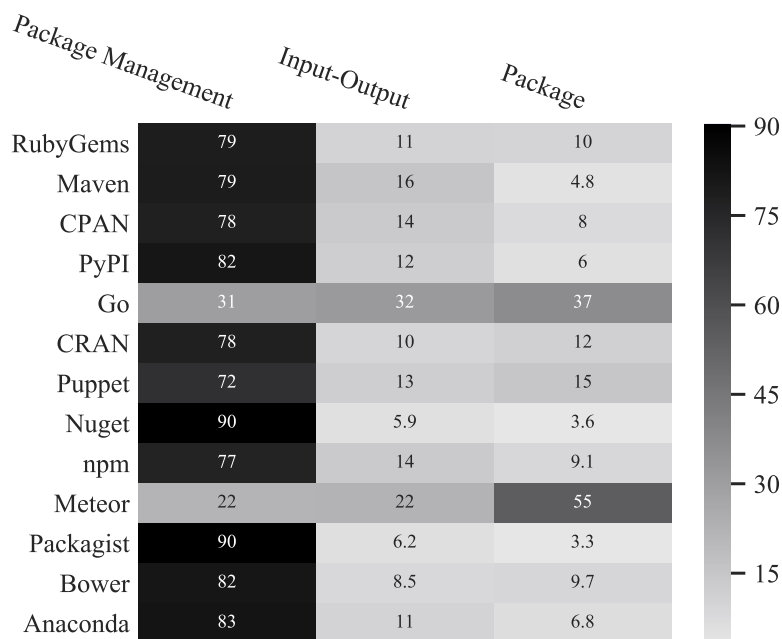


Figure 3.4: Package ecosystem vs. topic themes, where most developers of package ecosystems ask questions on package management except Go and Meteor

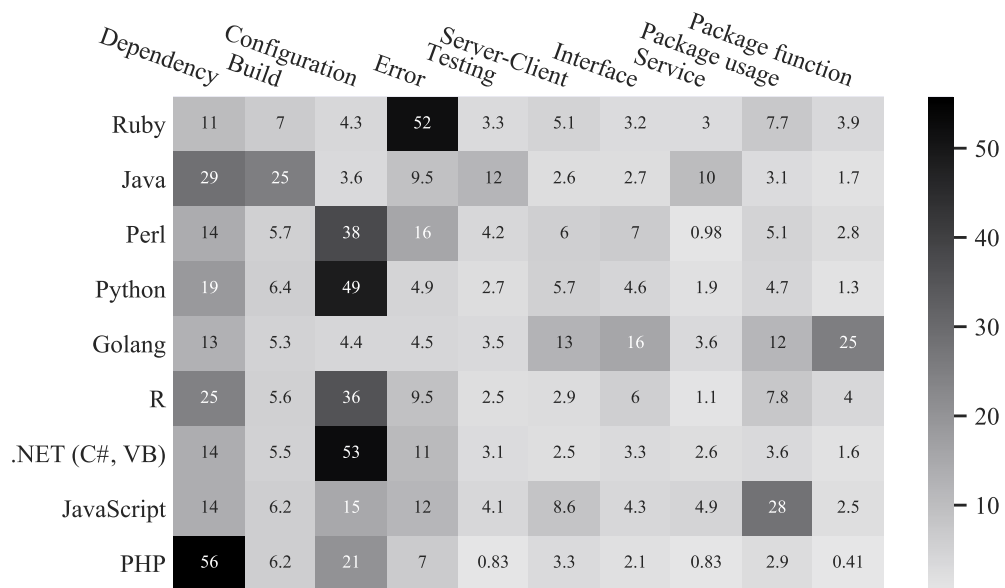


Figure 3.5: Language feature heatmap shows that different package ecosystems in different languages provide different developers experiences.

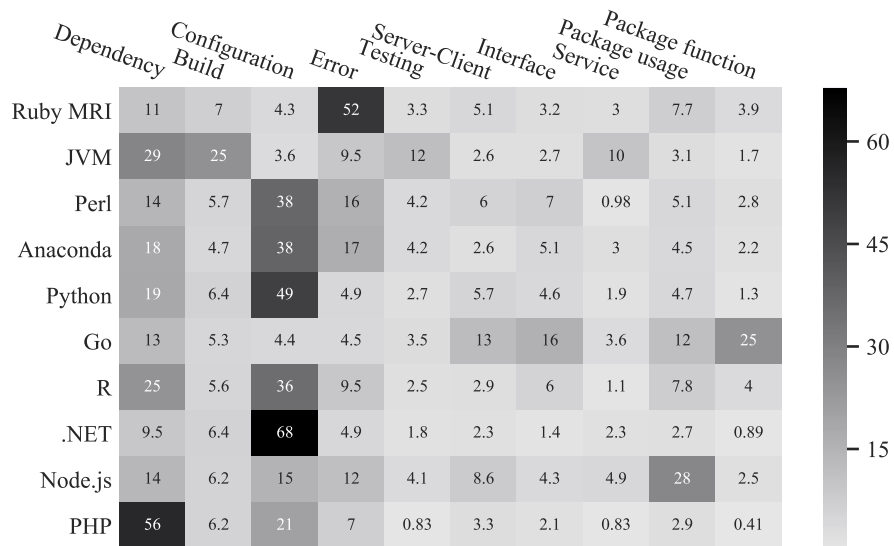


Figure 3.6: Environment feature heatmap shows that different package ecosystems in different environments provide different developers experiences.



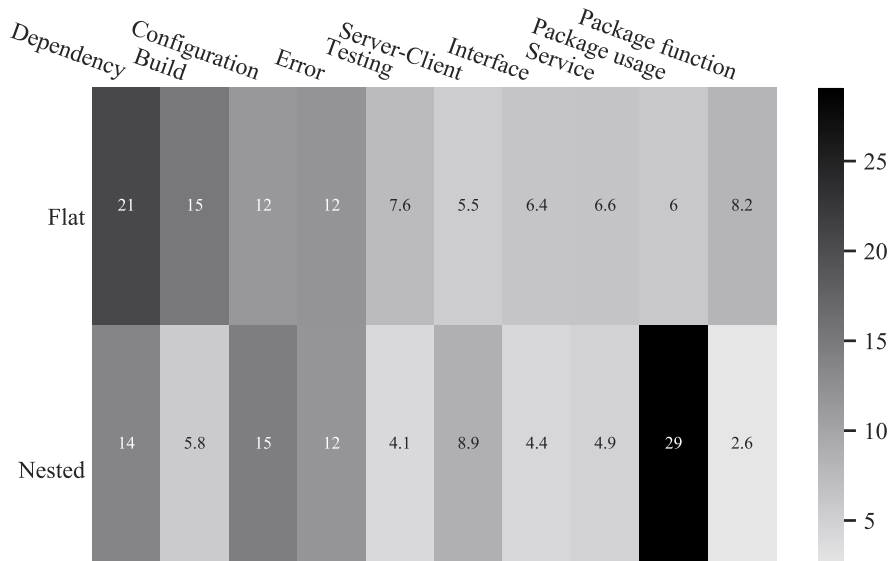


Figure 3.7: Dependency tree feature heatmap shows that dependency tree correlates with developers experiences (i.e., package ecosystems with a nested dependency face package usage issues, while flat dependency trees face questions related to dependencies)

### 3 Investigating Root Cause of Package Management Discussion and Their Kinds of Questions

In this section, I conducted a qualitative study of third-party package discussion on three package ecosystems to get deeper understanding of the questions asking and their underlying causes. From the curated 13 package ecosystem, I selected three popularly known package ecosystems npm, maven, and nuGet, etc.

Initially I was able to collect a total of 114,834 (i.e., maven:74,657 , npm:30,136 , and NuGet:10,041 question posts) stack overflow questions for these three package ecosystems. Afterward, I created representative sample, maintaining 95% confidence level and a confidence interval of 5 for each package ecosystem. This resulted in a total of 1131 question samples from three package ecosystem (i.e., maven:382, npm:379, and NuGet:370 posts). Once I obtained the question samples, I manually analyzed them to identify the question asking about and underlying causes of asking a package ecosystem question. To accomplish the goals,

I formulated two research questions. In the following, I explained each research question, motivation, approach, and the results.

### 3.1 What kind of questions do developers ask about third-party package management?

**Motivation.** After detail examination of the third-party package discussion topics, I aim to investigate the kind of questions developers asking. This analysis is important because it helps identify the nature of the challenges that arise during package management. In addition, question-type analysis helps software developers better understand the role of Stack Overflow.

**Approach.** Our approach to answer RQ 3.1 is a classification of question posts based on the coding scheme proposed by Treude et al. [91]. Details of the question coding scheme are described below:

- *How-to*: Posts that ask for instructions. For example: “Title: How to get latest version number of an artifact and replace it in target file? (Id: 26223226)” is a how-to type question asking for instruction to get latest version number of an artifact and replace it in target file.
- *Discrepancy*: Some unexpected behavior that the person asking on the third-party package post wants explained. For example: “Title: Spring Boot JPA & H2 Records Not Persisted (Id: 27843682)” is a discrepancy question asked for solution on persistence issues between spring boot JPA & H2 records.
- *Environment*: Posts about the environment either during development or after deployment. For example, “NPM Windows Path Problems (Id: 25120982)” is a question related to package management environment setting.
- *Error*: Posts that include a specific error message. For example: “Title: Error running Google App Engine quick start : POM for com.google.app engine:app engine-maven-plugin:jar:1.9.24 is missing (Id: 31576681)” is question asking for solution of a general error message related to google app engine.

- *Decision help*: Asking for an opinion. For example: “Can I invoke a local bean into a ear file from a Javax-WS into a war file- apache-tomee-plus-1.7.4 (Id: 51072906)” is question related to decision help.
- *Conceptual*: Posts that are abstract and do not have a concrete use case. For example: “Difference between mvn appengine:update and mvn appengine:deploy in Google App Engine (Id: 40094090)” is a conceptual question.
- *Review*: Posts that are either implicitly or explicitly asking for a review. For example: “Is there any possibility of deleting libraries stored maven central? (Id: 25133985)” is a question where developer asked for a review on the possibility of deleting libraries stored maven central.
- *Non-functional*: Third-party package posts about non-functional requirements such as performance or memory usage. For example: “Java project runs slow from JAR but fast from IDE (Id: 41861330)” is a question asked for solution on performance of Java project.
- *Novice*: Often explicitly states that the person belong third-party package posts is a novice. For example: “The mssing package org.spring framework.web (Id: 12603723)” is a novice question according to the description in the question body.
- *Novice/How to*: Posts that belong to a novice asking for step by step tutorials. For example: “Maven2 - POM configure issue in Windows (Id: 5087296)” is novice/how-to question according to the description of the question body.

To ensure a systematic method and reduce bias in the classification, I performed a Kappa agreement check using 30 random samples among three authors. Using the Kappa score calculator [96], I checked the agreement level and find score 86.67%, which was almost perfect. Confident with the agreement, two authors then continued to classify all samples through manual analysis of question title and body.

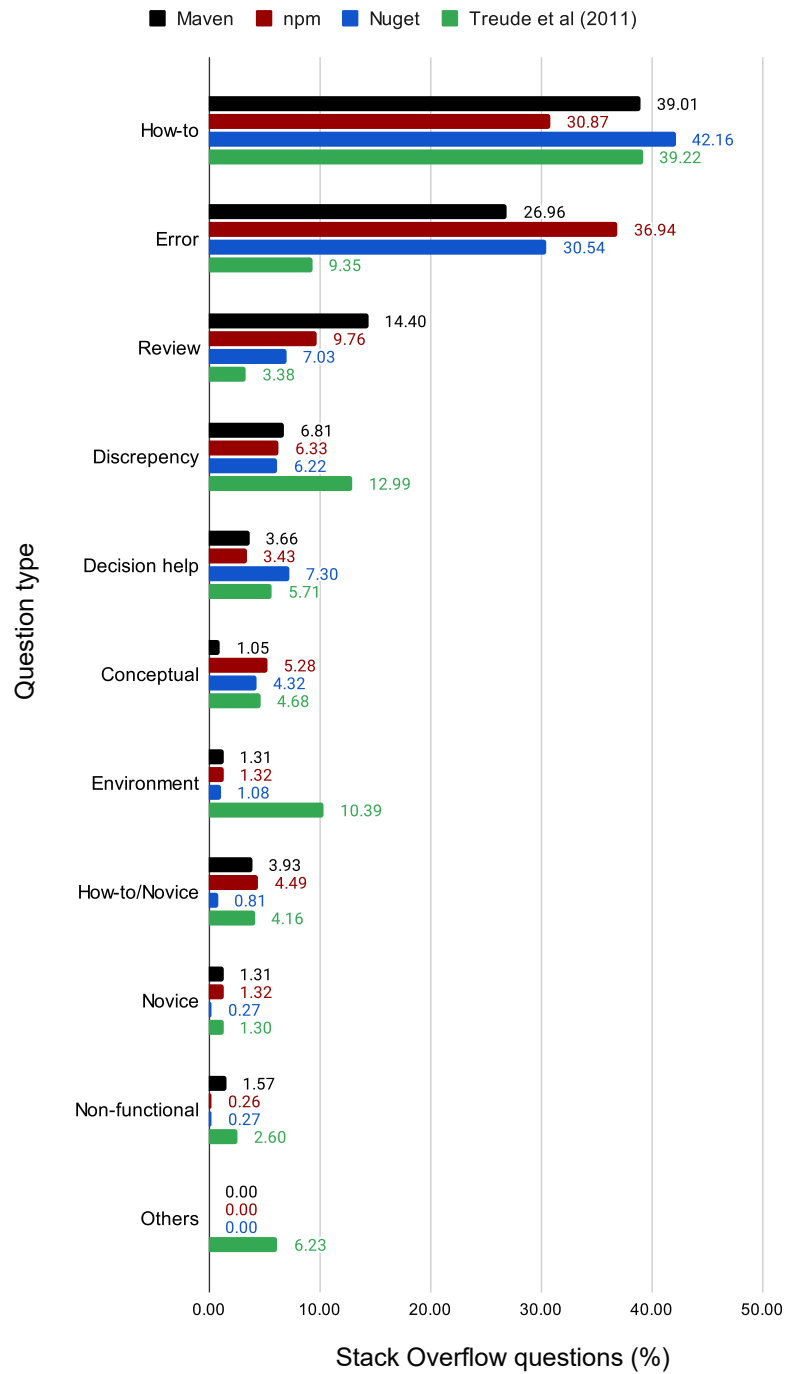


Figure 3.8: Percentage of third-party package posts by question coding compared to Treude et al. [91]. Result shows that ‘How-to’ and ‘Error’ messages are the most dominant questions asked.

**Results.** Figure 3.8, shows that most of the third-party package posts belong to **How-to** followed by **Error** message for three package ecosystems, confirming that developers are suffering from lack of instructions and intuitive error messages during third-party package management. In Maven, I find that **How-to** (39.01%) question is most dominant, followed by **Error** (26.96%) and **Review** (14.40%). Similar trend is also shared by NuGet, where **How-to** (42.16%) question is most dominant, followed by **Error** (30.54%) and **Decision help** (7.30%). On the other hand, in npm I find that **Error** (36.94%) message question is most dominant, followed by **How-to** (30.87%) and **Review** (9.76%).

In comparison of package ecosystem posts with the most generic results of Treude et al. [91], as shown in Figure 3.8, I find that **How-to** question is most dominant while **Error** message question is ranked as being the top type of question asked for developers of the npm ecosystem.

**Takeaway 5:** Third-party package related discussions are likely to arise from lack of instructions (i.e., around 31 to 42% of question samples) and error messages (i.e., around 27% to 37% of questions).

## 3.2 What are the underlying causes of questions related to third-party package management?

**Motivation.** Previous research showed that developers struggle to manage third-party package dependencies. However little is known of the underlying challenges related to using third-party packages from different packaging ecosystems. The target of this research questions is to deeper investigation of third-party package question posts. This analysis helps us to identify the underlying challenges faced by developers while using packages from different ecosystems.

**Approach.** Our approach to answer RQ 3.2 is to conduct a qualitative coding of the underlying causes of developers challenge faced while using a package from software packaging ecosystem. Similar to other work in software engineering by Hata et al. [39], I adopted an open coding strategy. First, I along with two authors independently sampled 30 questions to establish an initial set of causes. We then

added another 30 samples to make sure that no new causes appeared. To measure author agreement, I used the Kappa score [96]. After two rounds manual coding of 30 samples with three authors, we ended up with a Kappa score of 95.56% (almost perfect). Two author then coded the rest of the classification of third-party package posts. The four distinct underlying causes for third-party package posts are summarized below:

- *Specific migration*: Question posts related to dependency updates like an upgrade or downgrade to a specific version of the package, moving to new environment (language/OS/etc), incompatibility of a packages and other associated issues. Figure 3.9 shows that a developer struggle with an issues to migrate his project written in Java8 to Java9.

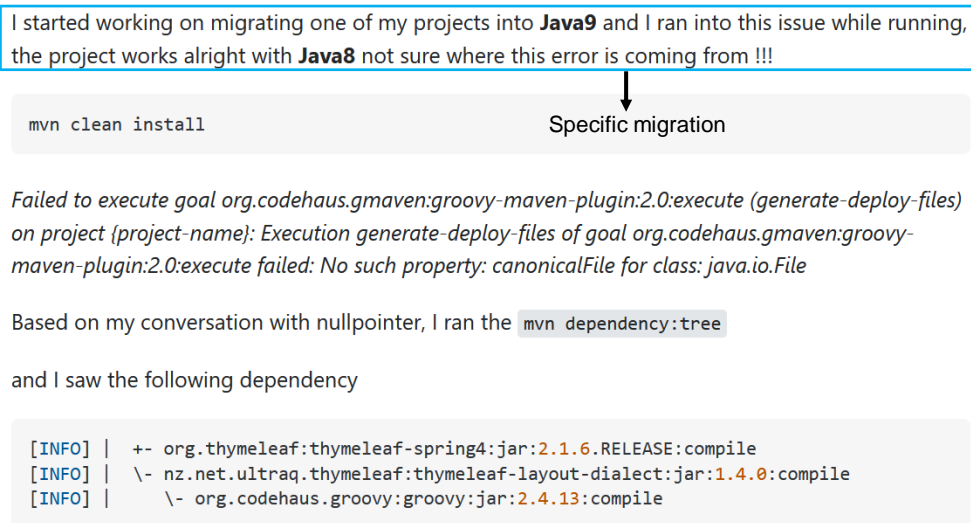


Figure 3.9: Post (Id: 50262939) that discuss issues related to specific migration

- *Package management tool usage*: Question posts related to technical details on package management systems such as installation, configuration of tools, and their associated costs. Figure 3.10 shows that a developer discuss about configuring the maven build tool to run his multi-module project in a custom way.

I have maven multi-modules project. At the parent level, i have some java files. And in the parent pom.xml, at the package phase i do some stuff.

Usually, when i run mvn package at parent level, the package phase of parent pom will be run and all the modules will be packaged as well.

I am looking for a way that allow me to do these (when i run mvn package):

- allow me to run only paren pom.xml (the script at the package phase), not the modules. This is the 1st priority.
- allow me to run paren pom.xml and some particular modules (like module 1, module 2 BUT not module 3 , module 4).

Can i use profile for those issue?

Thanks.

Package management tool usage

Figure 3.10: Post (Id: 4811870) that discuss issues related to package management tool usage

- *General dependency practices*: Question posts related to lack of technical knowledge on general dependency management practice, bugs, efficiency, etc. Figure 3.11 shows that, developers ask a question on the general ideas of maven packages dependency management.

In Maven, dependencies are usually set up like this:

```
<dependency>
  <groupId>wonderful-inc</groupId>
  <artifactId>dream-library</artifactId>
  <version>1.2.3</version>
</dependency>
```

Now, if you are working with libraries that have frequent releases, constantly updating the <version> tag can be somewhat annoying. Is there any way to tell Maven to always use the latest available version (from the repository)?

General ideas of dependency practice

Figure 3.11: Post (Id: 30571) that discuss issues related to general idea of dependency practice.

- *Others*: Posts that are tagged with a package ecosystem but do not fall in

the above three categories.

**Results.** Figure 3.12 shows that Package management tool usage is the most dominant underlying cause for all three package ecosystems, confirming that developers tend to report technical issues on Package management tool usage and not on specific migrations of dependencies. This finding is consistent with previous studies [2, 11, 26, 56, 66]. They reported that the complexity of software tools [66], installability of tools and packages [2, 11, 26, 56] are the root causes of the developers struggle during application development and maintenance. In detail, I find that Package management tool usage (72.25%) is most dominant underlying cause for Maven, followed by General dependency practices (10.73%). In npm, Package management tool usage (70.71%) is most dominant underlying cause, followed by General dependency practices (5.80%). In NuGet, Package management tool usage (63.51%) is most dominant underlying cause, followed by General dependency practices (18.65%) and Specific migration (13.51%).

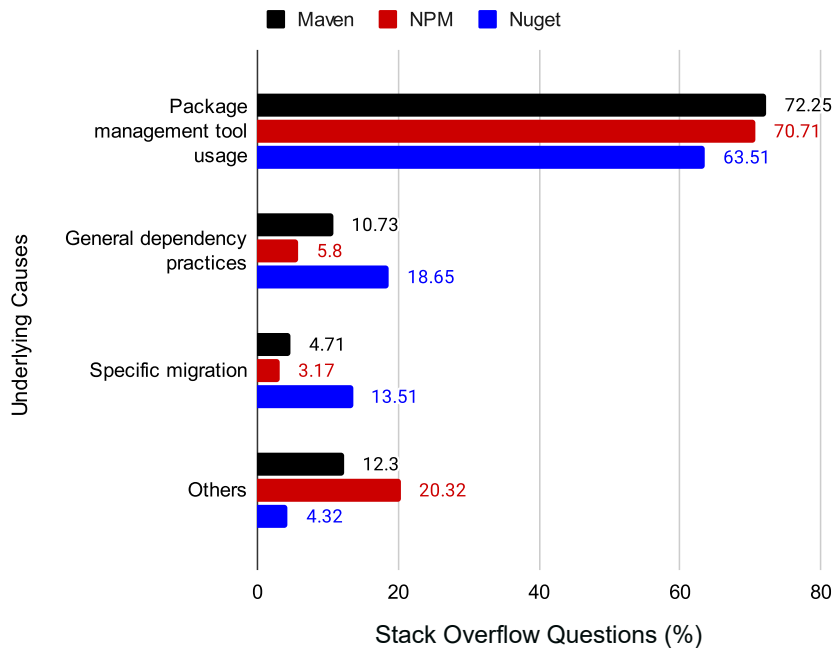


Figure 3.12: Percentage of third-party package posts by their underlying cause. I find that ‘*Package management tool usage*’ is the most dominant underlying cause.



**Takeaway 6:** The underlying causes of third-party package question posts can be categorized into package management tool usage, general dependency practices, specific migration, and others. Developers tend to ask questions that relate to `package management tool usage`, and not `specific migration`.

## 4 Implications

I now discuss implications of our work applicable to developers, and package manager designers:

**Developers** Developers should be conscious that their choice of a package ecosystem will impact their user experience – not all package ecosystems are the same as design choices such as hierarchy structure and language support are correlated with what questions developers are likely to encounter. When starting a new project, developers might be able to choose an ecosystem based on our insights. In other cases it might be too late to switch ecosystems, but developers might still be able to consider the benefits of (somewhat) compatible package ecosystems, such as npm and Meteor. Furthermore, as applications such as mobile apps require support for multiple technology stacks, developers should be aware of the trade-offs when switching technologies, i.e., when porting a Java application (Maven) to the web as a JavaScript application (npm). Developers can use our findings to better understand what technical background knowledge they should have regarding package ecosystems.

**Package manager designers** Designers need to be aware of the impact of package ecosystem design features on the problems that developers may encounter as some design choices are correlated with more questions on certain topics raised on Stack Overflow. Designers should be proactive about issues frequently encountered by package ecosystem users, by providing thorough documentation and/or improving package ecosystems where possible. Designers should also make it easy for developers to find the information they need to resolve issues, e.g., by pro-

viding good error messages, while other package ecosystems may require better documentation for package manager configuration.

## 5 Threats to Validity

This section describes the internal, external, and construct validity threats of this study.

**Internal Validity** - Threats to internal validity relate to experimental bias and error in conducting the analysis. The first threat is the accuracy of the methods used in this study. Especially choosing the appropriate number of topics ( $k=15$ ) for LDA model is considered as a potential threat. Labeling the topic based on keywords is another threat to the validity. To mitigate the threat, I and other two authors did a round-table style discussion and labelling of topics by manually reading top 30 posts of each topic and associated 20 keywords. This approach was also used by previous studies [5, 77]. Second, I perform manual analysis on random sample since the dataset size is large. To mitigate this challenge, I prepare representative samples for three package ecosystems, with a confidence level of 95% and a interval of 5. Thus, I believe that experimental bias and error in conducting the analysis were reduced.

**External Validity** - Threats to external validity relate to the generalizability of findings. In my study, I focused only on Stack Overflow which is the largest and most popular question-and-answer platform among developers. The findings of my study may not generalize to other question-and-answer platforms. However, my study is consistent with previous works that also utilized Stack Overflow data [5, 20, 77, 106, 107].

**Construct Validity** - Threats to construct validity are related to potential errors that can occur when extracting data about third-party packages related discussions for different software packaging ecosystems. The first construct threat is the validity of the collected data. I used Stack Overflow tags to identify posts related to the third-party package, but some posts may be incorrectly labeled (i.e., missing tags or incorrect tags). To reduce this threat, I created the list of

tags by following state-of-the-art approaches [5, 77].

The construct validity threats also relate to differences in theory, observations, and results. In the quantitative analysis, I compared developers topic of discussion on third-party packages for 13 packaging ecosystems where each ecosystem has different number of posts. For example, CRAN packaging ecosystem had small number of posts compared to other ecosystems. Such comparison may affect observations from the study results. However, our study is consistent with other previous works [9, 77, 100] that also compared different platforms with different number of posts.

In my qualitative analysis of classifying question types and underlying causes, the questions may be miscoded due to the subjective nature of our coding approach. To mitigate this threat, I took a systematic approach to validate the taxonomy and the comprehension understanding by the three authors in several rounds. Only until the Kappa score reaches 0.87 and 0.96, indicating that the agreement is almost perfect (0.81-1.00), we were able to complete the rest of the sample dataset.

## 6 Summary and Future Works

In this Chapter, I explore third-party package related discussion for thirteen package ecosystems to understand whether developers package management experience varies. The results show that developers questions related to third-party package can be clustered into ten different topics, which can be further grouped into three themes, and that different topics are prevalent for different ecosystems. In addition, I observe that most third-party package related questions arise from the lack of understanding different package management tool usage information rather than specific migration and general dependency practices.

The next logical step is further exploratory study to investigate the trade-offs between benefits and drawbacks of each package ecosystem will help piece together what the ideal package ecosystem should look like. Researchers could use our findings to prioritise research efforts, as our work is the first to acknowledge that developers encounter issues when using package ecosystems.

Future work is needed in particular in teasing apart the effects of different fea-

tures of package ecosystems and their interplay. While some features of a package ecosystem are given (e.g., the programming language), others provide more freedom to designers (e.g., dependency trees). In addition, interviewing, observing, and/or surveying software developers would further allow for triangulation of the findings to understand the impact of ecosystem features on developers' decision processes.

# 4 | Package Usage

## 1 Introduction

The purpose of this Chapter is to investigate developers experience (i.e., information needs) on package usage during software development. To accomplish this goal, I performed an exploratory study on package usage information from Stack Overflow in term of co-usage relationship. As defined by Todorov et. al [89], co-usage is the pattern of package dependencies that are used together. The rationale behind refining the co-usage relationship is to study problems caused by npm packages. In particular I investigate (i) whether I can detect package usage (i.e., co-usage) information from Stack Overflow and (ii) what the developers are looking for to solve problems related to the package. To address these, I study 2,100 Stack Overflow Q&A posts and matched them to 217,934 npm library packages. I reveal the following valuable lessons along the way:

- *Lesson 1:* I find that only three out of the top ten of the most used npm libraries are mentioned in Stack Overflow. The top-3 discussed npm libraries are `react`, `typescript`, and `webpack`. Again, the top-5 libraries that are less frequently discussed in Stack Overflow are `mocha`, `eslint`, `chai`, `babel-core`, and `lodash`. One possible reason is that, well-known libraries are well documented and may have their own forum, chat tools, etc. For this reason, there is no need to discuss them in Stack Overflow. Furthermore, I find that 87.95% of package co-usage mined from Stack Overflow exist in the latest npm package release.
- *Lesson 2:* Developers post answers provided with usage example or execute

command. Results do indicate the potential for a recommendation system, especially with the available execute commands and examples. Although Stack Overflow has been a useful resource for finding answers to questions, I find that popular and highly used libraries are not discussed as often. However, the accepted answers may prove useful, as I believe that the usage examples and executable commands could be reused for tool support.

The main contributions of this chapter are summarized as follows: (i) a quantitative study on the library usage information mined from Stack Overflow, and npm packages. (ii) a qualitative study on accepted answer posts of Stack Overflow to uncover developers useful responses to solve library usage issues.

The remainder of the chapter is organized as follows. Section 2 presents motivating example and research questions. Section 3 describes the data preparation. Section 4 presents the analysis approach. Section 5 reports the results for each research question. Section 6, discusses the implications from this study. Section 7 discloses the threats to validity of our study. Finally, I summarize this chapter in section 8 .

## 2 Motivating Example

Recent studies point out that Stack Overflow is a useful question-answering site among developers to communicate various issues [21, 54, 78, 90, 93]. In this paper, my motivation is to investigate the following assumptions:


- Package usage information mined from Stack Overflow is useful to solve developers issues while using libraries.
- Developer responses to package usage information in Stack Overflow follow some useful patterns that might be reused by the recommendation tools.

Figure 4.1 shows an example of a package co-usage related question post from Stack Overflow.<sup>1</sup> As shown in the figure, a developer posts a question on the error issue of `node_module` installation, resulting from two dependent packages

---

<sup>1</sup><https://stackoverflow.com/questions/46742824>

**Question**

 **babel-loader@7.1.2 requires a peer of webpack@2 || 3 but none was installed**  
 Asked 3 years, 7 months ago · Active 2 years, 4 months ago · Viewed 12k times

I am having this issue while installing all node\_modules. And this is making me crazy.

6

`babel-loader@7.1.2 requires a peer of webpack@2 || 3 but none was installed`

↓  
Throws Error

Here is my package.json file

```

{
  "name": "react-router-firebase-auth",
  "version": "0.1.0",
  "private": true,
  "devDependencies": {
    "babel-core": "^6.26.0",
    "babel-loader": "^7.1.2",
    "babel-plugin-transform-es2015-modules-commonjs": "^6.26.0",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "react-scripts": "0.9.5"
  },
}

```

Package co-usage example

I am using `create-react-app` for this project. So i could not change `webpack.config.js` file. What am i supposed to do here?

javascript reactjs npm babel-loader

**Accepted Answer**

Please read this post. It describes what a peer dependency is.

6 <https://stackoverflow.com/a/34645112/2379376>

What that means is that you have webpack not installed at all or you have a different version (webpack 1.x) installed. But this plugin needs webpack in version 2 or 3 to function properly.

What you can do is

```
npm install webpack -g
```

So npm will install the newest version of webpack on your system. But now other peer warnings could occur when other loaders need an older version of webpack.

Figure 4.1: A motivating example of npm package co-usage in Stack Overflow. The example shows that a developer encounters a issue when installing all node\_modules, due to two dependent packages babel-loader and webpack.

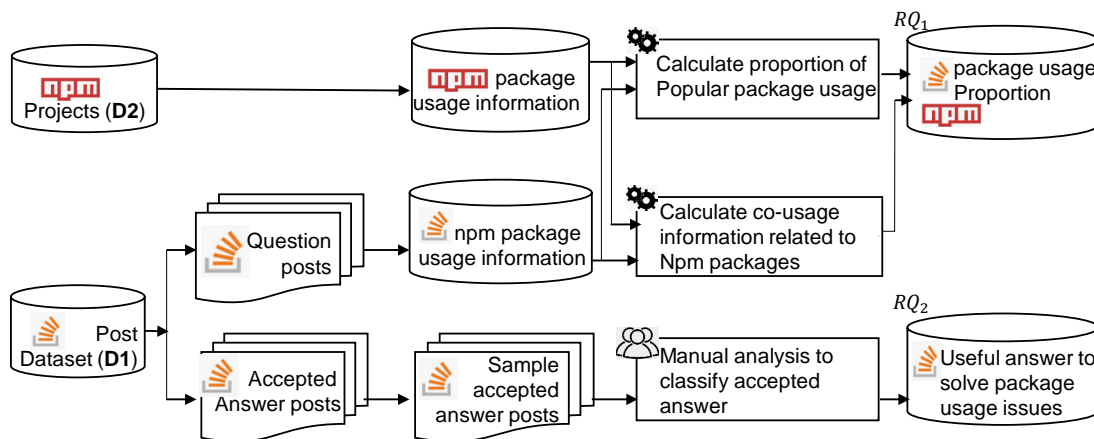


Figure 4.2: An overview of the methodology of our study.

`babel-loader` and `webpack`. A closer look into the question description, I observe that the successful installation of `babel-loader@7.1.2` requires the package dependency of `webpack@2||3`. This issue is solved by a simple installation command (i.e., `npm install webpack -g`) mentioned in the accepted answer of the question. Such a motivating example suggests that package usage information mined from question answering sites may help solve package-related issues.

**Research Questions:** Our goal in this chapter is to investigate whether package usage information mined from Stack Overflow can help maintain the packages. I formulate two research questions to guide this study.

- **RQ<sub>1</sub>: Package usage issues-** What proportion of package usage information mined from Stack Overflow exist in npm packages?

*Motivation.* Developers often share package usage information to communicate various software development issues through Stack Overflow. I would like to understand what is the difference in the package usage information between Stack Overflow and npm projects.

- **RQ<sub>2</sub>: Developers practices to solve package usage issues-** What kinds of answers are posted in response to questions that include package usage information?

*Motivation.* This research question investigates the developer’s response to



Table 4.1: Summary of dataset used in the study.

Data Source	Initial dataset	Final dataset
D1: Stack Overflow (npm question posts)	30,136	2,100
D2: libraries.io (npm projects)	100,5955	217,934

package usage information discussed in Stack Overflow posts. I argue that a closer look at the answers may reveal practical insights to improve real developers’ experience dealing with package co-usage issues.

### 3 Data Preparation

This study exclusively examines the npm package usage information from Stack Overflow. Stack Overflow is the largest and most popular question-answering site among developers, which allows them to ask developers and experts for development related questions. In addition, to compare with the package usage from the reality, I collect another dataset from the `libraries.io`.<sup>2</sup> `libraries.io` is popularly known to monitor package releases. Below, I describe the details of two studied datasets.

*(D1) from Stack Overflow posts:* I rely on the SOTorrent [12] to download the Stack Overflow data dump. The collection of npm related question posts was performed using a semi-automatic method similar to previous studies [77]. To do so, I first extract all tags from the question posts, and then I manually check whether or not the tags are directly related to the npm packages. After the examination, a list of eight tags that reflect npm packages posts are generated, i.e., ‘npm’, ‘npm-install’, ‘npm-script’, ‘npm-ignore’, ‘pnpm’, ‘npm-shrinkwrap’, ‘npm-start’, ‘npm-build’. I automatically identify all question posts using the defined tag list, resulting in 30,136 questions related to npm packages.

Next I further extract the npm related questions that contain the package usage information. I observed that several package names are as common as the natural language, i.e., `i`, `moment`, `should`, `express`, etc.). Thus, to reduce the bias, I only take those question posts that contain `package.json` files, resulting

<sup>2</sup><https://libraries.io/>

in 2,805 question posts. As I focus on the relatively popular libraries, I extract all packages from these question posts and sort out 628 npm packages whose frequency are greater than ten.

To ensure that the sample dataset contains most npm libraries, I use the cumulative sampling method to assure that question posts are saturated to cover all 628 npm packages. Finally, the Stack Overflow npm package usage dataset consists of 2,100 question posts, as shown in Table 4.1.

**(D2) from npm packages:** To compare the package usage information with Stack Overflow ones, I construct a dataset consisting of npm packages from `libraries.io`. To do so, I first downloaded the latest history data dump, which is available at <https://libraries.io/data>, resulting in a total number of 1,005,955 npm project release history.

Similar to the **(D1)**, I extract the libraries from these 1,005,955 projects and sort out 23,870 npm packages whose frequency are greater than ten. In the end, our `libraries.io` npm package usage dataset consists of 217,934 npm projects using the cumulative sampling method, as shown in Table 4.1.

## 4 Data Analysis

In this section, as shown in Fig. 2, I describe in detail the approaches used to answer two research questions.

### 4.1 Approach for RQ<sub>1</sub>: Package usage issues

I perform an exploratory study to understand to which extents do developers discuss the package usage information from Stack Overflow. Below, I describe the approach in detail.

**Proportion of popular package usage:** To analyze the proportion of package usage in Stack Overflow and npm projects, I extracted libraries from Stack Overflow posts obtained in datasets D1 and D2, separately. Afterward, I count and sort these packages based on the frequency.

**Co-usage information related to npm package:** To analyze the frequency of package usage information, I first need to identify npm package co-usage. To do so, I follow the two steps: (I) extract all the target packages

appearing in the code snippets from 2,100 Stack Overflow post related to npm package obtained in Section 3 (D1). Then, generate all possible binary combinations for co-usage of npm packages. For example, if a project contain three package dependencies ( $A, B, C$ ), then the generated list of binary package co-usage will be: ( $A, B$ ), ( $A, C$ ), ( $B, C$ ). After this step, I was able to retrieve 68,750 npm package co-usage information from Stack Overflow. (II) then I extract the npm package co-usage information based on the latest release, referring to 217,934 npm projects in Section 3 (D2). Finally, I check the occurrences of Stack Overflow npm package co-usage information in the generated package co-usage from latest npm projects using the following formula:  $\frac{\alpha}{\beta} \times 100$  where  $\alpha$ =Number of Stack Overflow npm package co-usage found in the latest npm project release, and  $\beta$ =Total npm package co-usage extracted from Stack Overflow.

In addition, to understand the issues raised by package related question, I extract the title information from 2,100 Stack Overflow posts obtained in D1 and automatically extracted the keywords using traditional Nature Language Processing (NLP) , including stop word removals. The output is a corpus of title keywords.

To visually understand the package related issues asked in the SO, I generate a word cloud based on the constructed title corpus.

*Observation 1- The npm post that discuss package usage information are mostly relate to different type of errors.* Figure. 4.3 shows the Word cloud based on Stack Overflow posts titles. The word cloud shows that npm posts regarding package usage information are primarily related to various types of errors like installation error, build failure, etc.

## 4.2 Approach for RQ<sub>2</sub>: Developers practices to solve package usage issues

I conduct a qualitative analysis to investigate the accepted answer post from Stack Overflow. I analyze the accepted answer since these answers are solutions that work for developers.<sup>3</sup> Below I describe the representative sample construction and the manual coding process.

---

<sup>3</sup><https://stackoverflow.com/tour>



computer software program, script, or command. As shown in Fig. 4.4, `npm install -g @angular/cli` is an execute command to install ‘angular/cli’.

- *Step by step Instruction:* The accepted answer contains step by step information to get the solution. In definition, instructions are detailed (i.e., step by step) information about how something should be done or operated. As shown in Fig. 4.4, the accepted answer contains three distinct step to solve the library usage issue (i.e., Angular CLI installation, generating and serving an Angular project, and open local host page in the browser.)
- *Usage Example:* The accepted answer explicitly mentions examples, referred to external links, source code, configuration files, etc. In definition, usage examples are defined as models, or typical cases (like external links, source code, etc.) used to solve a problem. As shown in Fig. 4.4, the external link mention in the beginning of the answer is usage example.

In the second round, to assure that there is no new cases, the three authors coded another 25 samples and I found that the initialized codes can totally fit. In the third round, to test the comprehension understanding, I coded another 20 samples and used the Kappa score to measure the agreement. The score is 0.82, which is implied as nearly perfect [96]. Based on this encouraging result, the first author then coded the rest of the representative sample independently.

## 5 Results

In this section, the results for each research question are described. First, I describe the result analysis, and then highlight our findings and answer each question.

### 5.1 Answering RQ<sub>1</sub>: Package usage issues

To show the proportion of popular package usage, I depict tables to statistically show the package usage between Stack Overflow and npm projects. Then, to analyze the frequency of Stack Overflow npm package co-usage in the latest npm

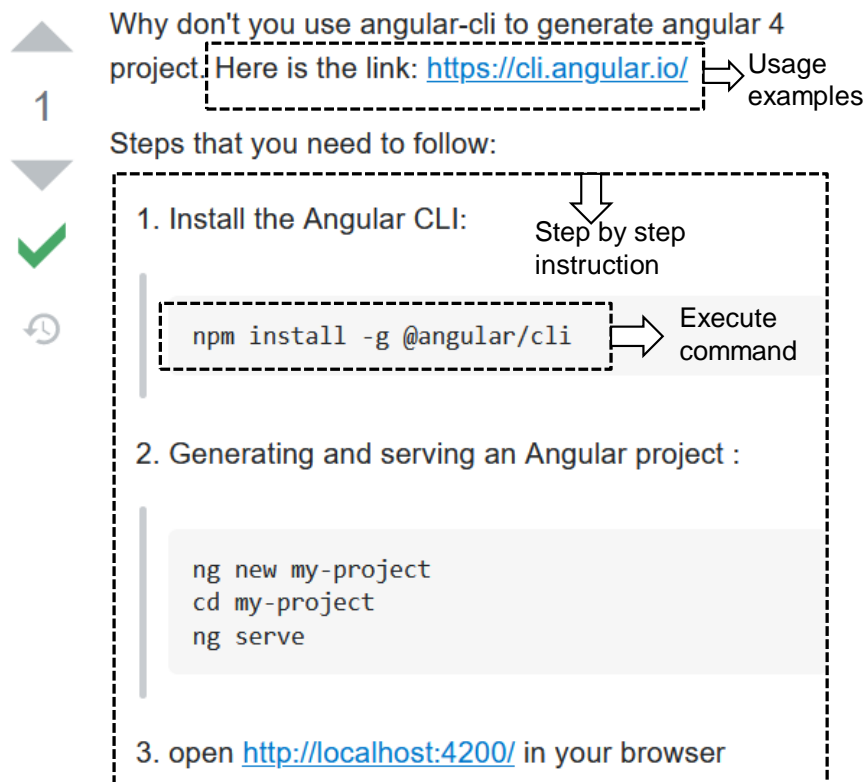


Figure 4.4: An example that motivates to classify developers response. In the answer I observe that, it contains usage examples, execute command, and step by step instruction.

Table 4.2: Top-15 npm packages extracted from Stack Overflow posts with their proportion and rank in the latest npm projects. Result shows that Only three out of top-10 npm packages are mostly discussed in Stack Overflow.

npm packages	Count (Stack Overflow)	Count (npm projects)	Rank (Stack Overflow)	Rank (npm projects)
react	586	42, 591	1	9
typescript	548	57,864	2	4
webpack	489	52,453	3	5
rxjs	471	12,339	4	69
zone.js	462	8,570	5	119
react-dom	461	32,941	6	16
@angular/core	434	10,050	7	95
@angular/common	434	9,406	8	103
@angular/compiler	426	9,170	9	110
@angular/platform-browser	424	8,482	10	120
@angular/platform-browser-dynamic	419	7,634	11	132
jquery	413	9,263	12	108
@angular/forms	401	6,608	13	147
@angular/http	388	5,433	14	169
@angular/router	380	5,583	15	166

projects, I calculated the ratio using formula (i.e.,  $\frac{\alpha}{\beta} \times 100$ ) discussed in the approach.

**Observation 2- Only three out of top-10 npm packages are mostly discussed in Stack Overflow.** Table 4.2 shows the top-15 packages discussed in Stack Overflow with their proportion and ranks in the latest npm projects. The top-3 discussed npm packages are `react`, `typescript`, and `webpack`. Again, Table 4.3 shows the top-15 package usage extracted from the latest npm projects. I observe that, the top-5 packages in the latest npm projects which are less frequently discussed in Stack Overflow are `mocha`, `eslint`, `chai`, `babel-core`, and `lodash`. One possible reason is that, such well-known libraries are well documented and may have their own forum, chat tools, etc. For this reason, there is no need to discuss them in Stack Overflow.

**Observation 3- 87.95% of the Stack Overflow package co-usage information exist in the latest npm project release.** Furthermore, Table 4.4 shows the top-15 Stack Overflow package co-usage mentioned by developers. I

Table 4.3: Top-15 package usage extracted from the latest npm projects with their proportion and rank in the Stack Overflow posts. The top package usage patterns from npm projects shows that application developers top usage packages are different from Stack Overflow.

npm package	Count (npm projects)	Count (Stack Overflow)	Rank (npm projects)	Rank (Stack Overflow)
mocha	101898	126	1	65
eslint	81767	199	2	45
chai	58368	114	3	78
typescript	57864	548	4	2
webpack	52453	489	5	3
babel-core	51351	309	6	26
lodash	45618	348	7	19
babel-loader	44398	340	8	22
react	42591	586	9	1
jest	41188	115	10	76
babel-eslint	39336	134	11	68
babel-cli	38038	102	12	83
eslint-plugin-import	36019	94	13	90
@types/node	35197	320	14	24
rimraf	34466	139	15	63

observed that most of the package co-usage mentioned by developers are related to `angular` followed by (`typescript`, `zone.js`). The top co-usage patterns from Stack Overflow and their rank hints that developers face most error type issues when they use angular packages.

**RQ<sub>1</sub> Summary:** Analysis result shows that, only three out of top-10 npm packages are mostly discussed in Stack Overflow. In addition, 87.95% of the Stack Overflow npm package co-usage information exist in the latest npm project release.



Table 4.4: Top-15 package co-usage extracted from Stack Overflow posts except `angular` since rest of the top co-usage are related to `angular`. The top co-usage patterns and their rank in Stack Overflow indicate that developers discuss most package dependency issues related to `angular` followed by (`'typescript'`, `'zone.js'`).

Package Co-usage	Rank	Count
( <code>'typescript'</code> , <code>'zone.js'</code> )	9	317
( <code>'zone.js'</code> , <code>'rxjs'</code> )	15	290
( <code>'react-dom'</code> , <code>'react'</code> )	17	288
( <code>'typescript'</code> , <code>'rxjs'</code> )	21	283
( <code>'karma'</code> , <code>'karma-jasmine'</code> )	31	258
( <code>'zone.js'</code> , <code>'core-js'</code> )	33	251
( <code>'core-js'</code> , <code>'rxjs'</code> )	39	233
( <code>'webpack'</code> , <code>'babel-loader'</code> )	40	230
( <code>'typescript'</code> , <code>'core-js'</code> )	40	230
( <code>'jasmine-core'</code> , <code>'karma-jasmine'</code> )	43	225
( <code>'karma-jasmine'</code> , <code>'karma-chrome-launcher'</code> )	44	223
( <code>'babel-core'</code> , <code>'babel-loader'</code> )	45	220
( <code>'typescript'</code> , <code>'tslint'</code> )	46	216
( <code>'typescript'</code> , <code>'karma'</code> )	47	210
( <code>'typescript'</code> , <code>'karma-jasmine'</code> )	48	209

## 5.2 Answering RQ<sub>2</sub>: Developers practices to solve package usage issues

To show the useful accepted answer attributes pattern in response to the package usage question by developers, I prepare all possible combinations for three manually curated attributes: Execute command, Step by step instruction, and Usage example, respectively. Thus, I obtain eight distinct combinations (i.e., subsets), including the others (i.e., empty set). Finally, I calculate the percentage of each variety in the manually analyzed representative sample.

*Observation 4- Our results show that, accepted answers posted by developers in response to questions that include package usage information mostly contain usage examples (i.e., includes real-life examples, external links, source code, build configuration files, etc.).* Figure 4.5 shows the analysis result of accepted answers posted by developers in response to questions that include package usage information. I observe that `usage example` (36.76%) is most dominant attribute in accepted answer, followed by `execute command` (19.58%). The third most frequent (15.03%) attribute combination in the accepted answer is `execute command` and `usage example`. These findings hint that application developers are suffering from a lack of technical depth in managing third-party libraries in their applications.

**RQ<sub>2</sub> Summary:** Result shows that 37.76% accepted answers posted by developers in response to questions that include package usage information contain `usage example` followed by `execute command` 19.58%.

## 6 Discussion

In order to aid application developers faced with package usage issues, I conducted an empirical study to understand the usefulness of package usage information mined from Stack Overflow. I learned two lessons along the way:

*Lesson 1-* Many of the library usage information on Stack Overflow is not from the popular npm package. I find that only three out of the top ten of the

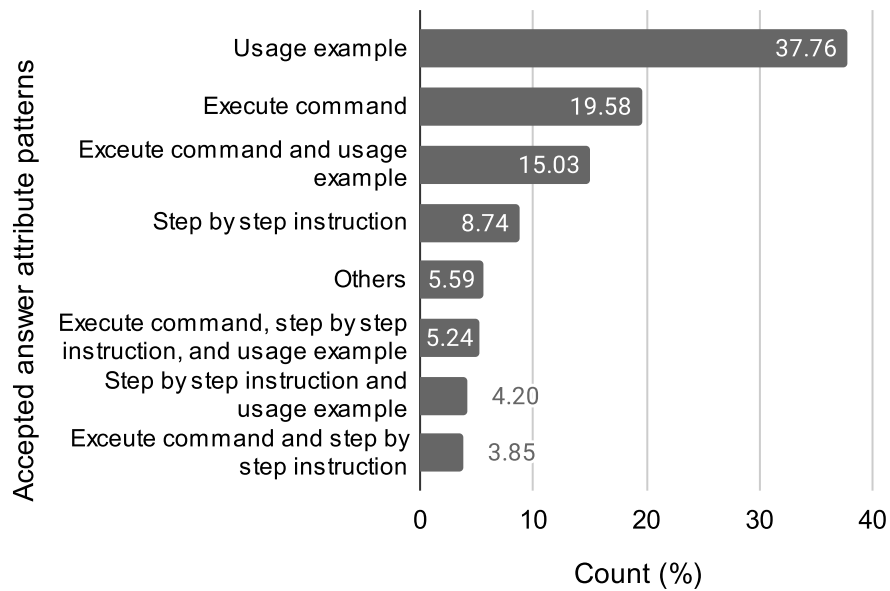


Figure 4.5: Analysis of accepted answers posted in response to questions that include package usage information. Result shows that 37.76% accepted answers contain `usage example` followed by `execute command` 19.58%.

most used npm libraries are mentioned in Stack Overflow. The top-3 discussed npm packages are `react`, `typescript`, and `webpack`. Again, the top-5 libraries that are less frequently discussed in Stack Overflow are `mocha`, `eslint`, `chai`, `babel-core`, and `lodash`. Furthermore, I find that 87.95% package co-usage mined from Stack Overflow exist in the latest npm package release. The npm post that discuss package usage information are mostly relate to different type of errors.

**Lesson 2-** Developers tend to post answers that are usage example or execute command. The good news is that maybe the answers can be useful for any npm developer that suffers from similar issues. There is potential for a recommendation system, especially with the available execute commands and examples available.

## 7 Threats to validity

This section describes the threats to validity that may affect our research.

**Internal Validity:** Threats to internal validity refer to experimental bias. In this study, I found two major internal threads that could affect the results. First, is the pre-processing of the dataset I decide the number of posts (2100) and npm packages (217,934) based on cumulative extraction of npm libraries and the generated co-usages. I continue the cumulative extraction until all the libraries and the co-usage cover. Second, in RQ<sub>2</sub> I perform manual analysis on random sample since the dataset size is large. To mitigate this challenge, I prepare representative sample consists of 286 randomly selected accepted answer, with a confidence level of 95% and a interval of 5.

**External validity:** Threats to external validity refer to the generalizability of our findings. My datasets consist of npm packages from libraries.io and Stack Overflow posts. Stack Overflow is a popular platform for question and answers from developers with various domains and experts. Hence, our observations and results can not be generalized for other package managers like Maven, NuGet, and others. Besides, I consider only those Stack Overflow posts that contain `package.json` file. Selecting more question posts may cause variation of top package co-usage results.

**Construct validity:** Threats to construct validity refers to the suitability of my evaluation measure. In my qualitative analysis of classifying accepted answers (RQ<sub>2</sub>), the answer patterns may be miscoded due to the subjective nature of our coding approach. To mitigate this threat, I took a systematic approach to validate the taxonomy and the comprehension understanding by the three authors in several rounds. Only until the Kappa score reaches 0.82, indicating that the agreement is almost perfect (0.81-1.00), we were able to complete the rest of the sample dataset.

## 8 Summary and Future Works

In this Chapter, I examine the usefulness of package usage information mined from Stack Overflow. I perform a case study on npm package co-usage information from Stack Overflow question posts (2100) and libraries.io (217,934 npm projects)

dataset. Although Stack Overflow has been a useful resource for finding answers to questions, I find that popular and highly used packages are not discussed as often. However, I can see that the accepted answers may prove useful, as I believe that the usage examples and executable commands could be reused or be used for tool support. In my future work, I will develop tool support that will utilize Stack Overflow usage examples and executable commands extracted from accepted answers to assist npm application developers.

## 5 | Conclusion

To understand the developer's experience (i.e., the challenges and information needs) of using third-party package from software packaging ecosystems, in this thesis I have investigated third-party package related discussions through popular question-and-answer site such as Stack Overflow. The main contributions of this thesis are divided into two parts. These are (i) package management and (ii) package usage. In the following, I summaries the results, implications, and possible future scopes of this thesis:

- *Developers experience in package management.* In this work, I explore third-party package related discussions to understand whether developers experience of package management varies in different software packaging ecosystems. The results show that developers questions related to third-party package can be clustered into ten different topics, which can be further grouped into three themes, and that different topics are prevalent for different ecosystems. In addition, I observe that most third-party package related questions arise from the lack of understanding package management tool usage information rather than specific migration and general dependency practice.
- *Developers experience in package usage.* In this work, I examine the usefulness of package usage information mined from Stack Overflow. Although Stack Overflow has been a useful resource for finding answers to questions, I find that popular and highly used packages are not discussed as often. However, I can see that the accepted answers may prove useful, as I believe that the usage examples and executable commands could be reused or be used for tool support.

In summary, the results of this thesis highlight the challenges and information needs associated with using third-party packages from software packaging ecosystem for developers, package manager designers and researchers.

## 1 Implications and Suggestions

The main goal of this thesis is to help developers and package manager designers in (I) identifying the challenges of using third-party packages from software packaging ecosystems and (II) understanding the information needs to solve package usage challenges. Thus, the empirical findings from this thesis are valuable both to developers, and package manager designers. Below, I summarize the suggestions for developers and package manager designers:

- *Developers:* The developers should consider the package management feature before choosing a software packing ecosystem because the findings of my study shows that specific features of package management has correlation with developers experience. They can optimize technology stack selection in software development where possible. For instance, Meteor facilitate complete software development with single programming language. In addition, it ensure development activity with less code, and easy to learn than npm. Moreover, developers should use popular packages in software development if there exist alternative choice. My study finding shows that popular packages have less issues and also have better chance to get supports from question-answering discussion sites.
- *Package manager designers:* The package manager designers should conduct online survey or face to face interview to understand how package management features impact developers decision process because not all packaging ecosystems have same language, environment and dependency tree structure. In addition, they should ensure that package management tools are easy to use for developers.

## 2 Opportunities for Future Research

There are still a lot of research aspect that can be done in order to help developers towards improving experience of using packages from software packaging ecosystems. Following are the research opportunities I see for the immediate future.

Researchers can perform an exploratory study to investigate the trade-offs between benefits and drawbacks of each package ecosystem that will help piece together what the ideal package ecosystem should look like. They can use our findings to prioritise research efforts, as our work is the first to acknowledge that developers encounter issues when using package ecosystems. Future work is also needed in particular in teasing apart the effects of different features of package ecosystems and their interplay. While some features of a package ecosystem are given (e.g., the programming language), others provide more freedom to designers (e.g., dependency trees). In addition, interviewing or surveying software developers would further allow for triangulation of the findings to understand the impact of ecosystem features on developers' decision processes. Researchers are also encouraged to investigate both positive and negative posts from social media such as Twitter and Facebook in order to gain a better understanding of developers' experiences with using third-party packages from different software packaging ecosystems. Additionally, developing recommendation tools (such as suggesting possible solutions, domain-specific experts, etc.) using resources from Stack Overflow and social media platforms may prove useful to developers in dealing with package usage challenges.



# References

- [1] Pietro Abate, Roberto DiCosmo, Ralf Treinen, and Stefano Zacchiroli. Mpm: a modular package manager. In Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering, pages 179–188, 2011.
- [2] Pietro Abate, Roberto Di Cosmo, Louis Gesbert, Fabrice Le Fessant, Ralf Treinen, and Stefano Zacchiroli. Mining component repositories for installability issues. In 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pages 24–33. IEEE, 2015.
- [3] Pietro Abate, Roberto Di Cosmo, Georgios Gousios, and Stefano Zacchiroli. Dependency solving is still hard, but we are getting better at it. In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 547–551. IEEE, 2020.
- [4] Rabe Abdalkareem. Reasons and drawbacks of using trivial npm packages: The developers’ perspective. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, page 1062–1064, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351058. doi: 10.1145/3106237.3121278. URL <https://doi.org/10.1145/3106237.3121278>.
- [5] Ahmad Abdellatif, Diego Costa, Khaled Badran, Rabe Abdalkareem, and Emad Shihab. Challenges in chatbot development: A study of stack overflow posts. In Proceedings of the 17th International Conference on Mining Software Repositories, pages 174–185, 2020.
- [6] Bram Adams, Herman Tromp, Kris De Schutter, and Wolfgang De Meuter. Design recovery and maintenance of build systems. In 2007 IEEE International Conference on Software Maintenance, pages 114–123. IEEE, 2007.

- [7] Amritanshu Agrawal, Wei Fu, and Tim Menzies. What is wrong with topic modeling? and how to fix it using search-based software engineering. Information and Software Technology, 98:74–88, 2018.
- [8] Md Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. Caps: a supervised technique for classifying stack overflow posts concerning api issues. Empirical Software Engineering, 25(2):1493–1532, 2020.
- [9] Miltiadis Allamanis and Charles Sutton. Why, when, and what: analyzing stack overflow questions by topic, type, and code. In 2013 10th Working Conference on Mining Software Repositories (MSR), pages 53–56. IEEE, 2013.
- [10] Kamel Alrashedy, Dhanush Dharmaretnam, Daniel M German, Venkatesh Srinivasan, and T Aaron Gulliver. Scc++: predicting the programming language of questions and snippets of stack overflow. Journal of Systems and Software, 162: 110505, 2020.
- [11] Josep Argelich and Inês Lynce. Cnf instances from the software package installation problem. In RCRA, 2008.
- [12] Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In Andy Zaidman, Yasutaka Kamei, and Emily Hill, editors, Proceedings of the 15th International Conference on Mining Software Repositories (MSR) 2018, pages 319–330. ACM, 2018. doi: 10.1145/3196398.3196430.
- [13] Sebastian Baltes, Lorik Dumani, Christoph Treude, and Stephan Diehl. Sotorrent: reconstructing and analyzing the evolution of stack overflow posts. In Andy Zaidman, Yasutaka Kamei, and Emily Hill, editors, Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018, pages 319–330. ACM, 2018.
- [14] Veronika Bauer, Lars Heinemann, and Florian Deissenboeck. A structured approach to assess third-party library usage. In 2012 28th IEEE International Conference on Software Maintenance (ICSM), pages 483–492. IEEE, 2012.
- [15] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. How the apache community upgrades dependencies: an evolutionary study. Empirical Software Engineering, 20(5):1275–1317, 2015.

- [16] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. Journal of machine Learning research, 3(Jan):993–1022, 2003.
- [17] Chris Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. When and how to make breaking changes: Policies and practices in 18 open source software ecosystems. ACM Transactions on Software Engineering and Methodology (TOSEM), 30(4):1–56, 2021.
- [18] Christopher Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. How to break an api: cost negotiation and community values in three software ecosystems. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pages 109–120, 2016.
- [19] Partha Chakraborty, Rifat Shahriyar, Anindya Iqbal, and Gias Uddin. How do developers discuss and support new programming languages in technical q&a site? an empirical study of go, swift, and rust in stack overflow. Information and Software Technology, 137:106603, 2021.
- [20] Preetha Chatterjee, Minji Kong, and Lori Pollock. Finding help with programming errors: An exploratory study of novice software engineers’ focus in stack overflow posts. Journal of Systems and Software, 159:110454, 2020.
- [21] Chunyang Chen and Zhenchang Xing. Similartech: automatically recommend analogical libraries across different programming languages. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pages 834–839, 2016.
- [22] Elena A Chernikova and Mikhail A Shalaev. Distributed linux build system for elbrus hardware platform. In 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIconRus), pages 256–258. IEEE, 2020.
- [23] Soyoung Choi and JooYoung Seo. An exploratory study of the research on caregiver depression: Using bibliometrics and lda topic modeling. Issues in Mental Health Nursing, pages 1–10, 2020.
- [24] J. Cox, E. Bouwers, M. Eekelen, and J. Visser. Measuring dependency freshness in software systems. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), pages 109–118, 2015.

- [25] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 conference on computer supported cooperative work, pages 1277–1286, 2012.
- [26] Alexandre Decan, Tom Mens, and Maëlick Claes. An empirical comparison of dependency issues in oss packaging ecosystems. In 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 2–12. IEEE, 2017.
- [27] Alexandre Decan, Tom Mens, and Eleni Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In Proceedings of the 15th International Conference on Mining Software Repositories, pages 181–191, 2018.
- [28] Alexandre Decan, Tom Mens, and Philippe Grosjean. An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Software Engineering, 24(1):381–416, 2019.
- [29] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 2187–2200, 2017.
- [30] Jens Dietrich, David Pearce, Jacob Stringer, Amjed Tahir, and Kelly Blincoe. Dependency versioning in the wild. In 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pages 349–359. IEEE, 2019.
- [31] Ahmad Diyanati, Behrooz Shahi Sheykhahmadloo, Seyed Mostafa Fakhrahmad, Mohammad Hadi Sadredini, and Mohammad Hassan Diyanati. A proposed approach to determining expertise level of stackoverflow programmers based on mining of user comments. Journal of Computer Languages, 61:101000, 2020.
- [32] Yujie Fan, Yiming Zhang, Shifu Hou, Lingwei Chen, Yanfang Ye, Chuan Shi, Liang Zhao, and Shouhuai Xu. idev: Enhancing social coding security by cross-platform user identification between github and stack overflow. In 28th International Joint Conference on Artificial Intelligence (IJCAI), 2019, 2019.

- [33] Eliakim Gama, Sávio Freire, Manoel Mendonça, Rodrigo O Spínola, Matheus Paixao, and Mariela I Cortés. Using stack overflow to assess technical debt identification on software projects. In Proceedings of the 34th Brazilian Symposium on Software Engineering, pages 730–739, 2020.
- [34] Zhipeng Gao, Xin Xia, David Lo, John Grundy, and Yuan-Fang Li. Code2que: A tool for improving question titles from mined code snippets in stack overflow. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1525–1529, 2021.
- [35] Daniel M German, Jesus M Gonzalez-Barahona, and Gregorio Robles. A model to understand the building and running inter-dependencies of software. In 14th Working Conference on Reverse Engineering (WCRE 2007), pages 140–149. IEEE, 2007.
- [36] Rishabh Gupta and P Krishna Reddy. Learning from gurus: Analysis and modeling of reopened questions on stack overflow. In Proceedings of the 3rd IKDD Conference on Data Science, 2016, pages 1–2, 2016.
- [37] Junxiao Han, Emad Shihab, Zhiyuan Wan, Shuiguang Deng, and Xin Xia. What do programmers discuss about deep learning frameworks. EMPIRICAL SOFTWARE ENGINEERING, 2020.
- [38] Mubin Ul Haque, Leonardo Horn Iwaya, and M Ali Babar. Challenges in docker development: A large-scale study using stack overflow. In Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pages 1–11, 2020.
- [39] Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 9.6 million links in source code comments: Purpose, evolution, and decay. In Proceedings of the 41st International Conference on Software Engineering, page 1211–1221, 2019.
- [40] Abbas Javan Jafari, Diego Elias Costa, Rabe Abdalkareem, Emad Shihab, and Nikolaos Tsantalis. Dependency smells in javascript projects. IEEE Transactions on Software Engineering, 2021.

- [41] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. Structure and evolution of package dependency networks. In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pages 102–112. IEEE, 2017.
- [42] Kanako Komiya, Yuji Abe, Hajime Morita, and Yoshiyuki Kotani. Question answering system using q & a site corpus query expansion and answer candidate evaluation. SpringerPlus, 2(1):1–11, 2013.
- [43] Raula Gaikovina Kula, Coen De Roover, Daniel M German, Takashi Ishio, and Katsuro Inoue. Modeling library dependencies and updates in large software repository universes. arXiv preprint arXiv:1709.04626, 2017.
- [44] Raula Gaikovina Kula, Coen De Roover, Daniel M German, Takashi Ishio, and Katsuro Inoue. A generalized model for visualizing library popularity, adoption, and diffusion within a software ecosystem. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 288–299. IEEE, 2018.
- [45] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. Do developers update their library dependencies? Empirical Software Engineering, 23(1):384–417, 2018.
- [46] T.C. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: the state of the practice. IEEE Software, 20:35–39, 2003. doi: 10.1109/MS.2003.1241364.
- [47] Menghao Li, Wei Wang, Pei Wang, Shuai Wang, Dinghao Wu, Jian Liu, Rui Xue, and Wei Huo. Libd: scalable and precise third-party library detection in android markets. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pages 335–346. IEEE, 2017.
- [48] Henry Lieberman and Christopher Fry. Bridging the gulf between code and behavior in programming. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 480–486, 1995.
- [49] Qiaoling Liu, Eugene Agichtein, Gideon Dror, Evgeniy Gabrilovich, Yoelle Maarek, Dan Pelleg, and Idan Szpektor. Predicting web searcher satisfaction with existing community-based answers. In Proceedings of the 34th international

ACM SIGIR conference on Research and development in Information Retrieval, pages 415–424, 2011.

- [50] Shuang Liu, Ru-Yuan Zhang, and Tomoko Kishimoto. Analysis and prospect of clinical psychology based on topic models: hot research topics and scientific trends in the latest decades. Psychology, Health & Medicine, pages 1–13, 2020.
- [51] Mircea Lungu. Towards reverse engineering software ecosystems. In 2008 IEEE International Conference on Software Maintenance, pages 428–431. IEEE, 2008.
- [52] Mircea Lungu, Romain Robbes, and Michele Lanza. Recovering inter-project dependencies in software ecosystems. In Proceedings of the IEEE/ACM international conference on Automated software engineering, pages 309–312, 2010.
- [53] Christian Macho, Shane McIntosh, and Martin Pinzger. Automatically repairing dependency-related build breakage. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 106–117. IEEE, 2018.
- [54] Sonal Mahajan, Negarsadat Abolhassani, and Mukul R Prasad. Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1052–1064, 2020.
- [55] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design lessons from the fastest q&a site in the west. In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 2857–2866, 2011.
- [56] Serghei Mangul, Thiago Mosqueiro, Richard J Abdill, Dat Duong, Keith Mitchell, Varuni Sarwal, Brian Hill, Jaqueline Brito, Russell Jared Littman, Benjamin Statz, et al. Challenges and recommendations to improve the installability and archival stability of omics computational tools. PLoS biology, 17(6):e3000333, 2019.
- [57] Konstantinos Manikas and Klaus Marius Hansen. Software ecosystems—a systematic literature review. Journal of Systems and Software, 86(5):1294–1306, 2013.

- [58] Andrew McCallum. Mallet. 2020.
- [59] Shane McIntosh, Bram Adams, and Ahmed E Hassan. The evolution of ant build systems. In 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pages 42–51. IEEE, 2010.
- [60] Shane McIntosh, Bram Adams, Thanh HD Nguyen, Yasutaka Kamei, and Ahmed E Hassan. An empirical study of build maintenance effort. In 2011 33rd International Conference on Software Engineering (ICSE), pages 141–150. IEEE, 2011.
- [61] Shane McIntosh, Meiyappan Nagappan, Bram Adams, Audris Mockus, and Ahmed E Hassan. A large-scale empirical study of the relationship between build technology and build maintenance. Empirical Software Engineering, 20(6):1587–1633, 2015.
- [62] Sarah Meldrum, Sherlock A Licorish, and Bastin Tony Roy Savarimuthu. Crowd-sourced knowledge on stack overflow: A systematic mapping study. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, pages 180–185, 2017.
- [63] Samim Mirhosseini and Chris Parnin. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 84–94. IEEE, 2017.
- [64] João Eduardo Montandon, Cristiano Politowski, Luciana Lourdes Silva, Marco Tulio Valente, Fabio Petrillo, and Yann-Gaël Guéhéneuc. What skills do it companies look for in new developers? a study with stack overflow jobs. Information and Software Technology, 129:106429, 2021.
- [65] Suhaib Mujahid, Rabe Abdalkareem, Emad Shihab, and Shane McIntosh. Using others’ tests to identify breaking updates. In Proceedings of the 17th International Conference on Mining Software Repositories, MSR ’20, page 466–476, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375177. doi: 10.1145/3379597.3387476. URL <https://doi.org/10.1145/3379597.3387476>.
- [66] Bhaveet Nagaria and Tracy Hall. How software developers mitigate their errors when developing code. IEEE Transactions on Software Engineering, 2020.



- [67] Kevin Kyung Nam, Mark S Ackerman, and Lada A Adamic. Questions in, knowledge in? a study of naver’s question answering community. In Proceedings of the SIGCHI conference on human factors in computing systems, pages 779–788, 2009.
- [68] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. What makes a good code example?: A study of programming q&a in stackoverflow. In 2012 28th IEEE International Conference on Software Maintenance (ICSM), pages 25–34. IEEE, 2012.
- [69] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. Crossrec: Supporting software developers by recommending third-party libraries. Journal of Systems and Software, 161:110460, 2020.
- [70] Chris Parnin and Christoph Treude. Measuring api documentation on the web. In Proceedings of the 2nd international workshop on Web 2.0 for software engineering, pages 25–30, 2011.
- [71] Ivan Pashchenko, Duc-Ly Vu, and Fabio Massacci. A qualitative study of dependency management and its security implications. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 1513–1531, 2020.
- [72] Steven Raemaekers, Arie Van Deursen, and Joost Visser. Measuring software library stability through historical version analysis. In 2012 28th IEEE International Conference on Software Maintenance (ICSM), pages 378–387. IEEE, 2012.
- [73] Romain Robbes, Mircea Lungu, and David Röthlisberger. How do developers react to api deprecation? the case of a smalltalk ecosystem. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pages 1–11, 2012.
- [74] Martin Robillard, Robert Walker, and Thomas Zimmermann. Recommendation systems for software engineering. IEEE software, 27(4):80–86, 2009.
- [75] Martin P Robillard and Robert DeLine. A field study of api learning obstacles. Empirical Software Engineering, 16(6):703–732, 2011.
- [76] Chat Room. Package manager. system, 10(23):46, 2020.

- [77] Christoffer Rosen and Emad Shihab. What are mobile developers asking about? a large scale study using stack overflow. Empirical Software Engineering, 21(3): 1192–1223, 2016.
- [78] Riccardo Rubei, Claudio Di Sipio, Phuong T Nguyen, Juri Di Rocco, and Davide Di Ruscio. Postfinder: Mining stack overflow posts to support software developers. Information and Software Technology, 127:106367, 2020.
- [79] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. On the reaction to deprecation of 25,357 clients of 4+ 1 popular java apis. In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 400–410. IEEE, 2016.
- [80] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. Programmers’ build errors: a case study (at google). In Proceedings of the 36th International Conference on Software Engineering, pages 724–734, 2014.
- [81] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. Does your configuration code smell? In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pages 189–200. IEEE, 2016.
- [82] Diomidis Spinellis. Package management systems. IEEE software, 29(2):84–86, 2012.
- [83] Margaret-Anne Storey, Alexey Zagalsky, Fernando Figueira Filho, Leif Singer, and Daniel M German. How social and communication channels shape and challenge a participatory culture in software development. IEEE Transactions on Software Engineering, 43(2):185–204, 2016.
- [84] Mohammad Tahaei, Kami Vaniea, and Naomi Saphra. Understanding privacy-related questions on stack overflow. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pages 1–14, 2020.
- [85] Youshuai Tan, Sijie Xu, Zhaowei Wang, Tao Zhang, Zhou Xu, and Xiapu Luo. Bug severity prediction using question-and-answer pairs from stack overflow. Journal of Systems and Software, 165:110567, 2020.
- [86] Mahdi Teimouri. bccp: an r package for life-testing and survival analysis. Computational Statistics, pages 1–21, 2021.

- [87] Cedric Teyton, Jean-Remy Falleri, and Xavier Blanc. Mining library migration graphs. In 2012 19th Working Conference on Reverse Engineering, pages 289–298. IEEE, 2012.
- [88] Ferdian Thung, David Lo, and Julia Lawall. Automated library recommendation. In 2013 20th Working conference on reverse engineering (WCRE), pages 182–191. IEEE, 2013.
- [89] Boris Todorov, Raula Gaikovina Kula, Takashi Ishio, and Katsuro Inoue. Sol mantra: Visualizing update opportunities based on library coexistence. In 2017 IEEE Working Conference on Software Visualization (VISSOFT), pages 129–133. IEEE, 2017.
- [90] Christoph Treude and Martin P Robillard. Augmenting api documentation with insights from stack overflow. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), pages 392–403. IEEE, 2016.
- [91] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web?(nier track). In Proceedings of the 33rd international conference on software engineering, pages 804–807, 2011.
- [92] Chris Tucker, David Shuffelton, Ranjit Jhala, and Sorin Lerner. Opium: Optimal package install/uninstall manager. In 29th International Conference on Software Engineering (ICSE’07), pages 178–188. IEEE, 2007.
- [93] Gias Uddin, Foutse Khomh, and Chanchal K Roy. Mining api usage scenarios from stack overflow. Information and Software Technology, 122:106277, 2020.
- [94] Gias Uddin, Fatima Sabir, Yann-Gaël Guéhéneuc, Omar Alam, and Foutse Khomh. An empirical study of iot topics in iot developer discussions on stack overflow. Empirical Software Engineering, 26(6):1–45, 2021.
- [95] Pradeep K Venkatesh, Shaohua Wang, Feng Zhang, Ying Zou, and Ahmed E Hassan. What do client developers concern when using web apis? an empirical study on developer forums and stack overflow. In 2016 IEEE International Conference on Web Services (ICWS), pages 131–138. IEEE, 2016.
- [96] Anthony J Viera, Joanne M Garrett, et al. Understanding interobserver agreement: the kappa statistic. Fam med, 37(5):360–363, 2005.

- [97] Liting Wang, Li Zhang, and Jing Jiang. Iea: an answerer recommendation approach on stack overflow. Science China Information Sciences, 62(11):1–19, 2019.
- [98] Naixuan Wang, Jian Cao, Qing Qi, Qi Gu, and Shiyu Qian. Sonas: A system to obtain insights on web apis from stack overflow. In CCF Conference on Computer Supported Cooperative Work and Social Computing, pages 499–514. Springer, 2020.
- [99] Shaowei Wang, David Lo, and Lingxiao Jiang. An empirical study on developer interactions in stackoverflow. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, pages 1019–1024, 2013.
- [100] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? Empirical Software Engineering, 24(2):637–673, 2019.
- [101] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. What security questions do developers ask? a large-scale study of stack overflow posts. Journal of Computer Science and Technology, 31(5):910–924, 2016.
- [102] Yuan Yao, Hanghang Tong, Tao Xie, Leman Akoglu, Feng Xu, and Jian Lu. Detecting high-quality posts in community question answering sites. Information Sciences, 302:70–82, 2015.
- [103] Yanfang Ye, Shifu Hou, Lingwei Chen, Xin Li, Liang Zhao, Shouhuai Xu, Jiabin Wang, and Qi Xiong. Icsd: An automatic system for insecure code snippet detection in stack overflow over heterogeneous information network. In Proceedings of the 34th Annual Computer Security Applications Conference, pages 542–552, 2018.
- [104] Haoxiang Zhang, Shaowei Wang, Tse-Hsun Chen, and Ahmed E Hassan. Reading answers on stack overflow: Not enough! IEEE Transactions on Software Engineering, 2019.
- [105] Haoxiang Zhang, Shaowei Wang, Heng Li, Tse-Hsun Peter Chen, and Ahmed E Hassan. A study of c/c++ code weaknesses on stack overflow. IEEE Transactions on Software Engineering, 2021.
- [106] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridayesh Rajan, and Miryung Kim. Are code examples on an online q&a forum reliable?: a study of

api misuse on stack overflow. In 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pages 886–896. IEEE, 2018.

- [107] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. An empirical study of common challenges in developing deep learning applications. In 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), pages 104–115. IEEE, 2019.