Liu D, Ren ZL, Long ZT *et al.* Mining design pattern use scenarios and related design pattern pairs: A case study on online posts. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 35(5): 963–979 Sept. 2020. DOI 10.1007/s11390-020-0407-4

Mining Design Pattern Use Scenarios and Related Design Pattern Pairs: A Case Study on Online Posts

Dong Liu[1], Zhi-Lei Ren[1,2,*], *Member*, *CCF*, *ACM*, Zhong-Tian Long[3], Guo-Jun Gao[1], and He Jiang[1,2], *Member*, *CCF*, *ACM*, *IEEE*

[1] *School of Software, Dalian University of Technology, Dalian 116024, China*

[2] *Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, Dalian 116000, China*

[3] *DUT-RU International School of Information Science & Engineering at DUT, Dalian University of Technology Dalian 116620, China*

E-mail: dongliu@mail.dlut.edu.cn; zren@dlut.edu.cn; longzhongtian@126.com; ggjgao@163.com jianghe@dlut.edu.cn

Received February 27, 2020; revised July 31, 2020.

Abstract In common design pattern collections, e.g., design pattern books, design patterns are documented with templates that consist of multiple attributes, such as intent, structure, and sample code. To adapt to modern developers, the depictions of design patterns, especially some specific attributes, should advance with the current programming technologies, for example, "known uses", which exemplifies the use scenarios of design patterns in practice, and "related patterns", which describes the relatedness between a design pattern and the others within a context. However, it is not easy to update the contents of these attributes manually due to the diversity of the programming technologies. To address this problem, in this work, we conducted a case study to mine design pattern use scenarios and related design pattern pairs from Stack Overflow posts to enrich the two attributes. We first extracted the question posts relevant to each design pattern by identifying the design pattern tags. Then, the topics of the posts were discovered by applying topic modeling techniques. Finally, by analyzing the topics specified for each design pattern, we detected 195 design pattern use scenarios and 70 related design pattern pairs, involving 61 design patterns totally. These findings are associated with a variety of popular software frameworks and programming techniques. They could complement the existing design pattern collections and help developers better acknowledge the usage and relatedness of design patterns in today's programming practice.

Keywords design pattern, software documentation, Stack Overflow, topic model

1 Introduction

Software design patterns aim to document the design knowledge of experienced developers as proven solutions so that the recurring design problems can be handled conveniently [1]. To depict design patterns, structured representations consisting of several specific attributes are usually adopted in various design pattern collections [2]. For example, a design pattern in the Gang of Four (GoF) book [3] is described with a template that consists of 13 attributes, e.g., "intent" states what issues the design pattern address, "structure" presents the classes in the design pattern using a class diagram, and "sample code" illustrates how to implement the design pattern with code fragments. The attributes used in different design pattern collections may carry the same meaning, despite having different names [2].

Although there exist a number of design pattern collections [2], their depictions may be outdated, especially for some specific attributes. Two typical examples are "known uses" and "related patterns" in the

Regular Paper

Special Section on Software Systems 2020

This work was supported by the National Key Research and Development Program of China under Grant No. 2018YFB1003903, and the National Natural Science Foundation of China under Grant Nos. 61722202 and 61772107.

*Corresponding Author

GoF book. The "known uses" attribute indicates what examples of usage of the design pattern can be found in real software systems. New use scenarios of design patterns may emerge in new programming techniques, tools, and platforms. This attribute is also adopted in some other design pattern collections, e.g., a pattern-oriented software architecture (POSA) book [4]. The "related patterns" attribute indicates which other design patterns are related to this one (also named "see also" in the POSA book). Similarly, the relatedness between design patterns may be implied by some practical scenarios. Some new software frameworks may contain new co-operations of design patterns, and consequently, new related design pattern pairs. The presenting "known uses" and "related patterns" of the classical design pattern collections, such as the GoF book which has been published for over two decades, can hardly involve the current programming technologies.

It is not an easy job to update the contents of these attributes timely due to the diversity of the technologies in today's software engineering field [5]. In the existing design pattern collections, these contents may be written by experts. But it is difficult for an expert to enumerate the usages of design patterns in a domain that he/she is not familiar with.

In contrast, the programming Question and Answer (Q&A) websites①② usually assemble numerous developers with various backgrounds to post their issues or problems with a wide range of aspects [5]. In these posts, the modern practice of design patterns in multiple domains is likely to be involved. Hence, it is a possible way to enrich the attributes of design patterns by leveraging the crowdsource knowledge of the online posts. With over 4 million participants and hundreds of thousands of posts per month, Stack Overflow is one of the largest programming Q&A websites and has been used as a knowledge repository in a variety of software engineering studies [6]. It makes an ideal source to provide the contents of these design pattern attributes.

In this work, we conducted a case study on Stack Overflow posts to mine design pattern use scenarios and related design pattern pairs to enrich the "known uses" attribute and the "related patterns" attribute, respectively. First, we identified a set of Stack Overflow tags, each of which represents a certain design pattern. Next, the question posts relevant to each of the design patterns were extracted via the identified tags. Then, we applied an advanced probabilistic model to the textual contents of the posts for topic modeling. Finally, we attempted to detect the use scenarios and related design patterns of each design pattern by analyzing the topics specified for each design pattern. The relevant data, code, and results are available online③.

By the case study, we obtained 195 design pattern use scenarios and 70 related design pattern pairs, in which 61 design patterns are involved. Compared with those in the existing design pattern collections, our findings are associated with a wide range of popular software frameworks, such as Ruby on Rails, Spring, and Object Relational Mapping, and programming domains, such as mobile development, machine learning, and web service. Moreover, as the questions on Stack Overflow are continuously posted, our results can be updated correspondingly. These findings could be reasonable supplementaries to the depictions of the existing design pattern collections and help developers better acknowledge the usage and relatedness of design patterns in modern programming practice.

Similar to our work, some other studies also applied topic models in summarizing software relevant documents, such as commit-log comments [7], bug reports [8], and online posts [5, 9]. They usually aimed to figure out what topics were discussed and provided overviews of these documents. Compared with these studies, the objective of our work is clearer. We focus on mining design pattern use scenarios and related design pattern pairs by leveraging the topics, and propose the measures to achieve it.

The main contributions of this paper are listed as follows.

1) We present a methodology of mining design pattern use scenarios and related design pattern pairs from the posts of programming Q&A websites. The outcomes can be closely entwined with the current programming technologies, thus valuable for modern developers.

2) We report a case study on Stack Overflow, in which 195 design pattern use scenarios are detected. We further make some observations about the characteristics of these use scenarios. These use scenarios could inspire developers for practical usages of design patterns. They could also serve as candidates for "known uses" of design pattern relevant documents.

①Stack Overflow. https://stackoverflow.com/, July 2020.

②Software Engineering. https://softwareengineering.stackexchange.com/, July 2020.

③https://github.com/WoodenHeadoo/design-pattern-attributes, July 2020.

3) We also show the 70 pairs of related design patterns detected by the case study and analyze the types of the relatedness. These pairs imply the relatedness of design patterns in practice. They could be references for developers or examples of "related patterns" for creating design pattern relevant documents.

The rest of this paper is organized as follows. We present the procedures of data extraction and topic modeling in Section 2. We elaborate the case study of detecting the design pattern use scenarios and the related design pattern pairs in Section 3 and Section 4, respectively. The potential threats to validity are discussed in Section 5. The studies related to our work are outlined in Section 6. Section 7 concludes the paper.

2 Case Study Setup

In this section, we present the setup of our case study. We first describe the details of the extraction of Stack Overflow posts, and then we explain how to find topics from the posts by leveraging the topic modeling technique. The overall framework is shown in Fig.1.

2.1 Data Extraction

In this subsection, we aim to extract the Stack Overflow posts relevant to each design pattern.

2.1.1 Data Collection

The Stack Overflow data dump is publicly available on archive.org④. In the case study, we collected all the 38 485 049 posts spanning from August 2008 to December 2017. The starting time corresponds to when the Stack Overflow site began to operate. The end time is set to keep a time interval by the date we started the study so that the posts are relatively stable, i.e., with no more deletions or modifications [10].

2.1.2 Tags Identification

To extract design-pattern-relevant posts, we used the tags of Stack Overflow posts as the anchors [11–13]. A tag of a post is a label or keyword that summarizes and categorizes the post. A user is encouraged to attach one to five tags when posting a question in Stack Overflow. In this study, we aim to identify each tag that represents a certain design pattern. The following steps detail the procedure of tag identification.

First, we made a checklist of design patterns. The previous study [14] has constructed a design pattern catalog consisting of 425 design patterns collected from multiple sources. A design pattern may have multiple well-known names (aliases), for example, "model view controller" and "mvc" represent the same design pattern. We took this design pattern catalog⑤ as the checklist for identifying design pattern relevant tags.

Next, we extracted all the tags of the posts and investigated whether they could match with the design patterns in the checklist. A tag is a phrase consisting of several words that are separated by hyphens. A design pattern name is also a phrase with spaces or hyphens as delimiters. We deleted all the delimiters in

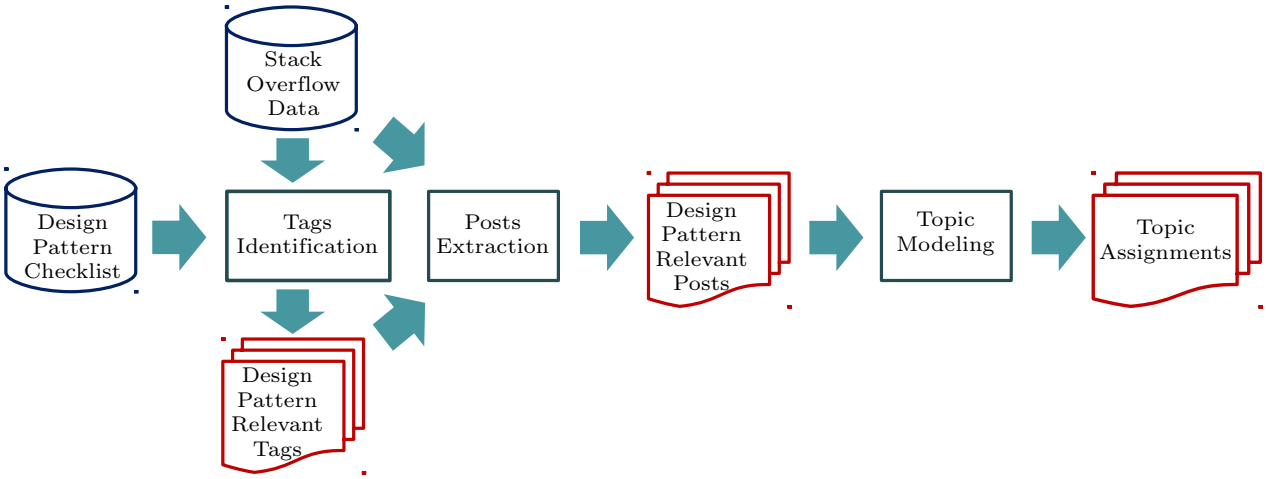

Fig.1. Overall framework to carry out our case study.

④https://archive.org/details/stackexchange, July 2020.

⑤https://github.com/WoodenHeadoo/design-pattern-catalog/wiki/design-pattern-catalog, July 2020.

the tags and design pattern names or aliases. We say a tag matches with a design pattern if the processed tag matches one of the processed names (aliases) of the design pattern. For example, tags "abstract-factory" and "activerecord" match design patterns "abstract factory" and "active record", respectively. Tag "data-access-object" and tag "dao" both match with "data access object" design pattern as "dao" is an alias. If a tag ends with "pattern", then the suffix "pattern" should be ignored. For example, tag "visitor-pattern" also matches with "visitor" design pattern.

Then we manually reviewed the matching tags to ensure accuracy. For each tag matching with a design pattern, if there existed a tag description in Stack Overflow, we reviewed it to clarify whether the tag really represented the design pattern in meaning. The false-positive matching tags were discarded.

At last, multiple tags were considered as a same tag if they all matched with a same design pattern. Totally, 94 unique tags (correspondingly 94 design patterns) were identified.

2.1.3 *Posts Extraction*

With the identified tags, we extracted the design-pattern-relevant posts. A question post was relevant to a design pattern if it contains the tag matching with the design pattern. As a question post is allowed to contain multiple tags, it may be relevant to multiple design patterns. Particularly, only question posts were adopted in this study, since an answer usually shares a same scenario with the question. Moreover, we discarded the design patterns involving less than 100 question posts as there may be too less information about the use scenarios or related patterns of them. Finally, our extracted data involved 175 213 question posts relevant to 61 design patterns. The design patterns and the corresponding Stack Overflow tags are shown in Table 1.

2.2 Topic Modeling

To conduct the case study, we should understand the contents of the design pattern relevant posts. However, the amount of the extracted posts is so large that it is difficult to review each one manually. Thus, we achieve the goals by leveraging an advanced topic modeling technique.

2.2.1 *Data Preprocessing*

The textual content of a question post includes the title and the body, and we merged them into a single

Table 1. Design Patterns with Matching Stack Overflow Tags

ID	Design Pattern	Tag
1	Abstract factory	abstract-factory
2	Active record	activerecord
3	Adapter	adapter
4	Bridge	bridge
5	Builder	builder, builder-pattern
6	CQRS	cqrs
7	Command	command-pattern
8	Composite	composite
9	Content negotiation	content-negotiation
10	CRTP	CRTP, f-bounded-polymorphism
11	DAO	data-access-object, dao
12	Data mapper	datamapper
13	DTO	data-transfer-objects, dto
14	Decorator	decorator
15	DI	dependency-injection
16	Domain model	domain-model
17	DCL	double-checked-locking
18	Event sourcing	event-sourcing
19	Facade	facade
20	Factory method	factory-method
21	Factory	factory-pattern, factory, factories
22	Federated identity	federated-identity
23	File transfer	file-transfer
24	Front controller	front-controller
25	Future	future
26	HMVC	HMVC
27	Interceptor	interceptor
28	Iterator	iterator
29	Lazy loading	lazy-loading, lazyload
30	Master/slave	master-slave
31	Materialized view	materialized-views
32	Mediator	mediator
33	Message broker	messagebroker
34	Messaging	messaging
35	MVC	model-view-controller
36	MVP	MVP
37	MVVM	MVVM
38	Object ID	objectid
39	Object pool	object-pooling, objectpool
40	Observer	observer-pattern
41	Page objects	pageobjects
42	Pipeline	pipeline
43	Pooling	pooling
44	Post/redirect/get	post-redirect-get
45	Publish/subscribe	publish-subscribe
46	Reactor	reactor
47	Record set	recordset
48	Reflection	reflection
49	Repository	repository-pattern
50	Service layers	service-layer
51	Service locator	service-locator
52	Sharding	sharding
53	STI	single-table-inheritance
54	Singleton	singleton
55	State	state
56	Strategy	strategy-pattern
57	Throttling	throttling, throttle
58	Unit of work	unit-of-work
59	Value object	value-objects
60	Viewcontroller	viewcontroller
61	Visitor	visitor, visitor-pattern

Note: CQRS = command and query responsibility segregation, CRTP = curiously recurring template pattern, DAO = data access object, DTO = data transfer object, DI = dependency injection, DCL = double checked locking, HMVC = hierarchical model view controller, MVC = model view controller, MVP = model view presenter, MVVM = model view view model, and STI = single table inheritance.

text. In this way, a post can be regarded as a document and topic modeling was carried out on the corpus of posts. Before that, we preprocessed the texts to remove irrelevant information.

At first, we eliminated the code snippets (enclosed in <code> or <pre> tag in Stack Overflow) as they are usually too short to contain meaningful contents [5, 15]. Next, we removed English-language stop words such as "the", "is", and "of", as they are not likely to lead to understandable topics [5]. The stop word list we used is provided by MySQL for full-text queries⑥. Besides, the HTML tags (e.g., <p>, <a href="...">), numbers, punctuation marks, and other non-alphabetic characters were discarded as they are less informative. Then, we used the Snowball stemmer⑦ to map each word in the text to its root form, such as "developing" to "develop" and "softwares" to "software", since the meanings of these forms are quite similar. At last, we counted the frequency of each remaining word and deleted the words occurring in less than five posts to filter out noisy terms.

After the preprocessing, we obtained a vocabulary containing 16 422 words.

2.2.2 *Topic Model Deployment*

Among various types of topic models, we choose the collapsed Gibbs Sampling algorithm for the Dirichlet Multinomial Mixture model (GSDMM) [16] in the case study, as it is more suitable for our purpose. Like some other probabilistic topic models, e.g., Latent Dirichlet Allocation (LDA) [17], GSDMM provides the probability distribution of words for each extracted topic so that we can comprehend a topic easily with the high probability words as the reference. In addition, GSDMM is also an efficient text clustering algorithm that assigns each post a specific topic. Therefore, we can understand the topic further by exploring some example posts corresponding to the topic.

Generally, GSDMM is a generative model that assumes the documents (posts) are generated based on some probabilistic distributions. Assuming the number of documents, the number of topics, and the vocabulary size are D, T, and V respectively, the generation contains three steps. At first, GSDMM generates a topic distribution vector $\boldsymbol{\theta}$ and a word distribution vector $\boldsymbol{\phi}_t$ ($t = 1, 2, \cdots, T$) for each topic. Next, a topic z_d is assigned to the d-th document ($d = 1, 2, \cdots, D$) according to the topic distribution $\boldsymbol{\theta}$. At last, each word in the d-th document is generated based on the corresponding word distribution $\boldsymbol{\phi}_{z_d}$ ($d = 1, 2, \cdots, D$).

Instead of applying the GSDMM model to the entire posts directly, we adapted it to involve the factor of design pattern. Specifically, we distributed the preprocessed posts to K categories according to their relevant design patterns ($K = 61$ in this study). A post can be in multiple categories simultaneously as it may contain multiple design pattern tags. Then, GSDMM was applied to each category. We should note that each category has its own topic distribution vector $\boldsymbol{\theta}_k$ ($k = 1, 2, \cdots, K$), but all the categories share a same batch of topics and word distribution vectors $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \cdots, \boldsymbol{\phi}_T$.

The reasons why we consider this setup are twofold. On the one hand, retaining a specific distribution of topics for each design pattern is more accurate to depict the design patterns than leveraging a unique topic distribution. On the other hand, it is more efficient and significant to share topics than to separate the posts of each design pattern totally, e.g., we may detect the usage of different design patterns in a same scenario.

To use GSDMM in practice, the parameters should be specified, i.e., α, the hyper-parameter for the topic distribution, β, the hyper-parameter for the word distribution, n, the number of iterations for Gibbs Sampling, and T, the number of topics. For α, β, and n, we adopted the default settings in [16] that $\alpha = 0.1$, $\beta = 0.1$, and $n = 150$. As the number of topics, T, is a key parameter that directly influences the outcomes of the case study, we determined it experimentally.

As mentioned before, GSDMM is also a clustering algorithm that assigns a specific topic for each post. According to the empirical results in [16], the number of clusters (actually assigned topics) grows first and finally gets stable near the ground truth when T is enlarged. We repeated this experiment on our post corpus by increasing T from 50 to 500 and recording the number of found clusters for each configuration. Fig.2 reports the mean results of eight trials for each value of T. As shown in Fig.2, the stable number of clusters is roughly 120. Therefore, we set $T = 120$ so that the true clusters tend to be found and few topics are likely to be redundant. With this number of topics, we got 115 clusters finally.

⑥https://dev.mysql.com/doc/refman/5.5/en/fulltext-stopwords.html, July 2020.

⑦https://www.nltk.org/api/nltk.stem.html, July 2020.

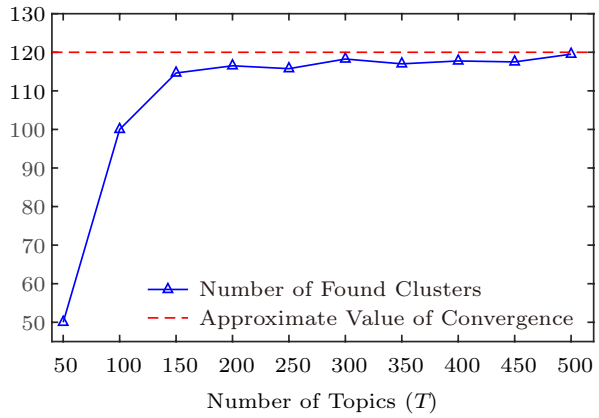

Fig.2. Number of clusters found by GSDMM with different values of T.

2.2.3 *Topic Model Outputs*

By inferring through the Gibbs Sampling algorithm [16], the topic assignment to each post (z_d, $d = 1, 2, \cdots, D$) was obtained. Moreover, GSDMM outputs two matrices that represent the distribution vectors, namely the pattern-by-topic ($K \times T$) matrix $\boldsymbol{\Theta}$, in which the element $\boldsymbol{\theta}_{kt}$ represents the probability for a post of the k-th design pattern to be assigned with the t-th topic, and the topic-by-word ($T \times V$) matrix $\boldsymbol{\Phi}$, in which $\boldsymbol{\phi}_{tv}$ denotes the probability for a post of the t-th topic to be generated by the v-th word. The sum of each row of $\boldsymbol{\Theta}$ or $\boldsymbol{\Phi}$ is 1.

These outputs can assist to perform the case study conveniently. With the pattern-by-topic matrix, we can acknowledge which topics are the emphases when considering a specific design pattern. In Fig.3, we visualize the pattern-by-topic matrix with a heat map. As presented in Fig.3, the posts relevant to a design pattern usually concentrate on some certain topics. Most other topics have low probabilities on this design pattern. Therefore, we need only to focus on these topics for efficiency when analyzing this design pattern.

With the topic-by-word matrix, we can comprehend a topic by reviewing the words of high probability. For instance, we can infer that the topic #49 (topic ID in the original order) is about Android development according to top words "java android view active layout app". Furthermore, with the topic assignments, we can choose a set of posts assigned with some topic as examples to help the comprehension.

3 Case Study Part 1: Use Scenarios of Design Patterns

In this section, we are to investigate the following research question.

*RQ*1. What are the possible use scenarios of each design pattern?

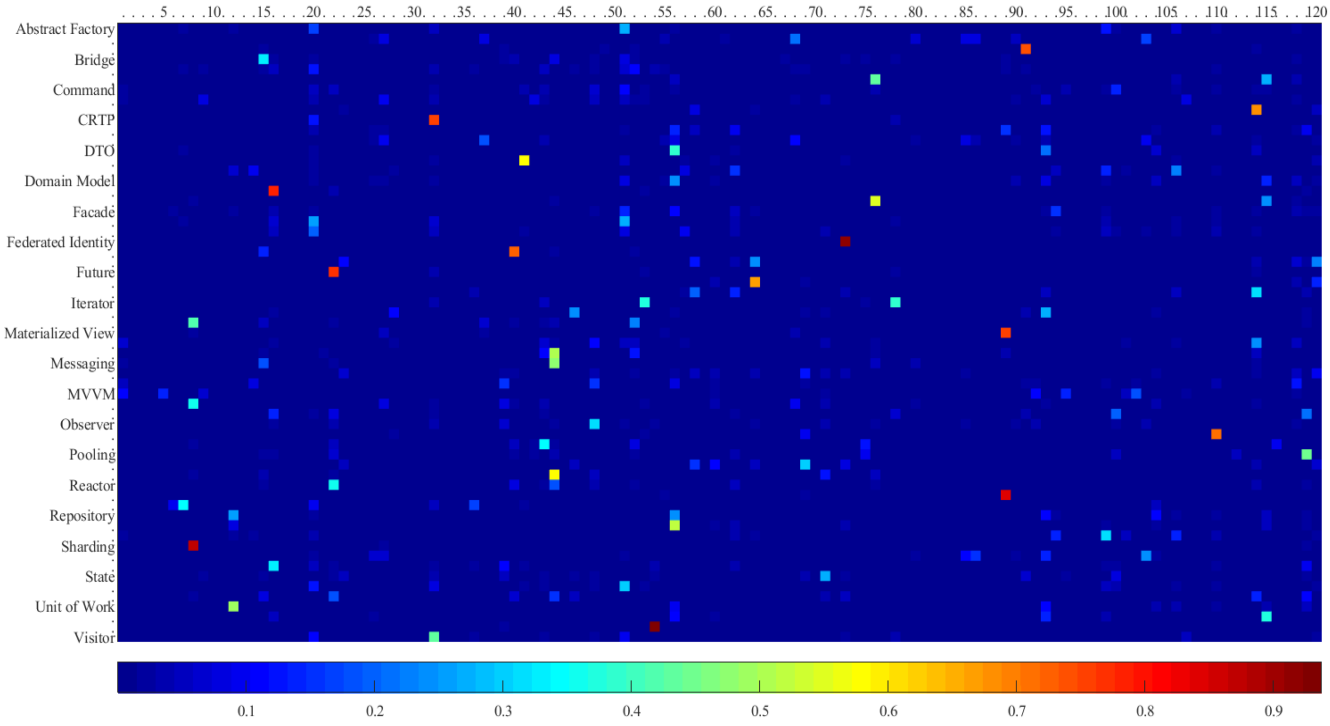

Fig.3. Distribution of topics on each design pattern ($\boldsymbol{\Theta}$).

3.1 Motivation

The topics of the Stack Overflow posts have been extracted and the posts tagged with a given design pattern tend to concentrate on some certain topics. That means, these topics, namely dominant topics, arouse wider concerns than other topics. Therefore, the dominant topics may imply some customary practical uses of the design pattern. In this part of the case study, we aim to detect the design pattern use scenarios from the dominant topics. These use scenarios, which correspond to the "known uses" attribute of the GoF book [3], could be potential materials for learning design patterns.

3.2 Approach

At first, we determined the dominant topics for each of the design patterns. For each design pattern, we sorted the topics according to the probabilities (the corresponding row of the pattern-by-topic matrix $\boldsymbol{\Theta}$) in descending order. As presented in Fig.4, the sorted probability distribution usually exhibits a long tail phenomenon. By connecting the tops of the bars into a curve, we detected the knee point [18] of the curve. The topics whose corresponding probabilities are greater than that at the knee point are the dominant topics.

Then, we manually analyzed each dominant topic to infer the use scenario behind it. For each dominant topic of a design pattern, we randomly sampled 10 representative posts that were assigned to the topic and involved the design pattern as a tag. Moreover, we extracted the top 15 words (keywords) with the highest probabilities according to the topic-by-word matrix $\boldsymbol{\Phi}$ to interpret the topic.

With the representative posts and the keywords, we attempted to conclude the use scenario of the design pattern. Three of the authors (the first, the third, and the fourth) participated in the use scenario summarizing process. Before that, the participants went over the related materials to get familiar with the involving design pattern. During the process, each participant independently reviewed the dominant topic and tried to summarize the implied use scenario with some phrases. As an ideally summarized use scenario should represent some common use of the design pattern, the texts containing these phrases and the design pattern were searched online to check whether such a use scenario could be verified by other sources. After all the participants completed the work, we discussed each one's summary together to make a compromise about the depictions of the use scenario.

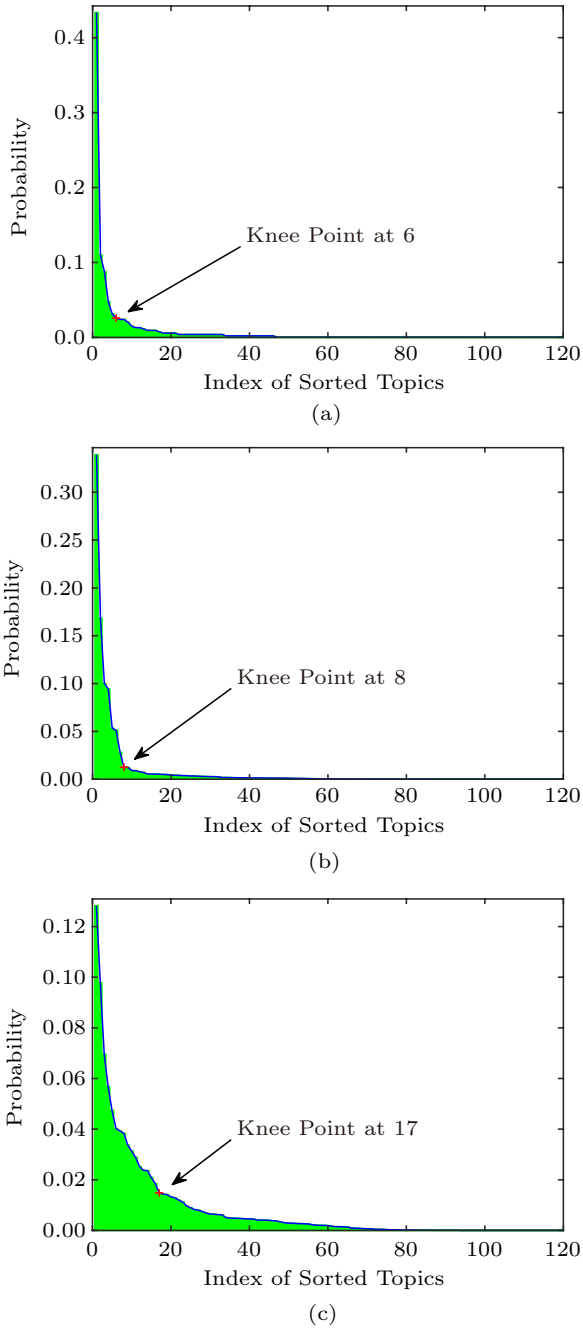

Fig.4. Sorted probability distribution of topics on each design pattern. (a) Visitor. (b) Reflection. (c) MVC.

3.3 Results

By reviewing the dominant topics of each design pattern manually, the use scenarios of these design patterns were detected. During this process, the scenarios behind multiple topics might appear to be quite similar for some design patterns. In this case, these topics were

merged into one use scenario. Moreover, some topics were discarded as they are not related to use scenarios, e.g., they are about how to implement some design patterns. Totally, we obtained 195 use scenarios that involve a wide range of frameworks, techniques, and domains. A part of the use scenarios are shown in Table 2. The full list is available online[8]. For each use scenario, a brief summary, the corresponding design pattern, and the corresponding topic ID(s) are presented. To enhance readability, we have provided online materials[8] for each use scenario. The online materials include official documents of software frameworks, articles of programming communities, and posts of technical blogs, in which these use scenarios are detailedly discussed.

Table 2. Use Scenarios of Design Patterns

Design Pattern Name	Topic ID	Summary	Online Materials[8]
Active record	68	Embedded in Ruby on Rails as ActiveRecord	u1
	80	PostgreSQL specific usage of active record	u2
	120	Modified version of active record pattern in CodeIgniter	u3
Command	100	Command pattern in game programming	u4.1, u4.2
	1, 48, 95	Command pattern in GUI design (WPF, GWT, WinForms)	u5.1, u5.2, u5.3
Composite	9	Composite in GUI design (SWT, GWT, WPF, JSF)	u6.1, u6.2, u6.3, u6.4
	93	Composite data entity	u7
	71	Composite in JavaScript frameworks (Marionette.js, ExtJS)	u8.1, u8.2
	43	Composite for file systems	u9
Lazy loading	93	Entity lazy loading in ORM (Entity Framework, Nhibernate)	u10.1, u10.2
	46	Images lazy loading in jQuery	u11.1, u11.2
	28	Feature modules lazy loading in Angular	u12
	91	Images lazy loading in Android ListView	u13
	69	DataTables lazy loading in PrimeFaces	u14
Master/slave	8	MongoDB master-slave replication (database)	u15
	52	Jenkins master/slave architecture (project management)	u16
	37	MySQL master-slave replication (database)	u17
	44	Using in ActiveMQ for high availability (message queue)	u18
	15	Bluetooth master-slave model (communication)	u19
	22	Master-slave programming paradigm in parallel computing	u20
MVVM	102	Constructing architectures in WPF	u21
	101	MVVM Light Toolkit	u22
	71	MVVM in KnockoutJS	u23
	39	MVVM in mobile development (Android, iOS)	u24.1, u24.2
	69	Applying MVVM to Kendo UI	u25
Object pool	100	Avoiding memory fragmentation in game programming	u26
	91	Reusing ListViews in Android development	u27
Pipeline	41, 43	Shell script pipeline (Unix shell, Powershell, Bash)	u28.1, u28.2, u28.3
	75	Pipeline for machine learning (Scikit-Learn)	u29
	116	Pipeline in MIPS architecture	u30
	52	Pipeline for projects Continuous Integration/Delivery (Jenkins)	u31
	42	Graphics pipeline (OpenGL, DirectX)	u32.1, u32.2
	44	Pipeline for web service (NServiceBus, Redis, BizTalk)	u33.1, u33.2, u33.3
	8	Pipeline for data processing (MongoDB, Hadoop)	u34.1, u34.2
	23	Pipeline for JavaScript and CSS assets	u35
Sharding	8	Database sharding (MongoDB)	u36
	27, 119	Database sharding (MySQL, PostgreSQL)	u37.1, u37.2
	44	Message queue sharding (RabbitMQ)	u38
	76	Akka cluster sharding	u39
Singleton	119	Singleton class for database connection (bad design)	u40
	62	Spring Bean singleton scope	u41
	54	Singleton UIViewController in iOS development (bad design)	u42.1, u42.2
	41	Singleton-decorator in Python	u43
State	71	React component state	u44
	100	Using state pattern in game programming	u45
	23	State pattern for routing in Angular UI.Router	u46
Visitor	107	Combining with the traversal strategies of tree structure	u47.1, u47.2
	36	Parsing abstract syntax trees	u48.1, u48.2
	43, 60	Transforming structures into XML files	u49

[8]https://github.com/WoodenHeadoo/design-pattern-attributes/blob/master/supplements.pdf, July 2020.

Counting all these scenarios, 59 design patterns and 72 topics are covered. The number of detected use scenarios for each design pattern ranges from 1 to 8. There are three design patterns that achieve the maximum number of use scenarios. They are "Dependency Injection", "Iterator", and "Pipeline". Some other design patterns also have relatively large numbers of use scenarios, such as "Interceptor" (7), "Decorator" (6), "Master/Slave" (6), "MVC" (6), and "Repository" (6). That means they are active design patterns in Stack Overflow. However, for the two design patterns, i.e., "Bridge" and "Strategy", we did not find significant use scenarios corresponding to them. The relevant question posts are mainly about the implementations or basic concepts of these two design patterns. Among the 72 topics, the topic #44 which is about message queues and the topic #93 which is relevant to data entities, are the most frequent ones that all participate in 10 use scenarios.

Furthermore, we analyze these use scenarios and make some observations.

• *A design pattern may be embedded or commonly applied into some specific software frameworks and become a key concept in these frameworks.* For example, "Active Record" is a design pattern that encapsulates both the database access and the domain logic. This design pattern is adopted in Ruby on Rails framework as a module named ActiveRecord. Several mechanisms of "Active Record", e.g., association, migration, and validation, are provided by Ruby on Rails as built-in features (u1). The "MVVM" design pattern is another example. "MVVM" usually serves as the core architecture when developing a Windows Presentation Foundation (WPF) application (u21). Some core features of WPF, including data binding, data template, and the resource system, make "MVVM" a suitable design pattern for this framework.

• *A design pattern may be used to handle some specific programming tasks as paradigms.* For instance, with the "Visitor" design pattern, we can perform operations on tree structures, especially parse abstract syntax trees (ASTs). By defining specific operations on a visiting node when traversing an AST, various tasks can be implemented, such as error recovery and binding resolution. Several code analysing tools have adopted this design pattern, e.g., Eclipse AST View (u48.1) and ANTLR (u48.2). Similarly, the "Composite" design pattern can be used to design graphical user interface applications. "Composite" is suitable for constructing complex user interface components by integrating simple controls, such as buttons, textboxes, and checkboxes. Some user interface development platforms, e.g., Google Web Toolkit (u6.2) and Java Server Faces (u6.4), leverage "Composite" to organize their applications.

• *Multiple design patterns may be applied to a same domain but with different perspectives.* For example, in the game programming domain, we detect three relevant design patterns, namely "Command", "Object Pool", and "State". When playing a game, the "Command" design pattern allows the players to configure the keyboard or mouse settings according to their preferences (u4.1, u4.2). During the game, the "State" design pattern makes a role change its actions accordingly when the state changes (u45). To improve the performance, "Object Pool" maintains a pool of reusable objects to avoid memory fragmentation (u26). Likewise, "Master/Slave" and "Sharding" are both design patterns aiming to achieve better performance and scalability of databases. The former separates the read and write operations by database replication (u15, u17) and the latter breaks a large database into smaller ones with nothing shared (u36, u37.1, u37.2).

• *A design pattern may have multiple use scenarios with distinct contexts.* For instance, "Lazy Loading" is a design pattern to delay the initialization of an object until needed. On the one hand, "Lazy Loading" can be used by a user interface for loading images (u11.1, u11.2). Specifically, an image will not be visible until the user scrolls down the page and the image placeholder comes into viewport. On the other hand, this design pattern can also be deployed on application data, i.e., deterring the loading of the data from database until the SQL query is triggered (u10.1, u10.2). Another example is the "Pipeline" design pattern which builds a workflow by assembling a series of actions. It has been used to perform machine learning tasks (Scikit-Learn, u29), build asynchronous messaging (NServiceBus, u33.1), and configure the jobs to build/test/deploy projects (u31), etc. The Unix shell also adopts the concept of "Pipeline" (u28.1).

• *Some idiomatic usages of design patterns are regarded as bad practice.* Inconsistent with our imaginations, some of the use scenarios are given negative comments by the answers in Stack Overflow. They are all about the "Singleton" design pattern. Specifically, the two use scenarios, i.e., making the database connection object as a "Singleton" (u40) and making the UIViewController in iOS as a "Singleton" (u42.1, u42.2), are not recommended in practice, as the use of singletons

may easily cause problems such as tight coupling between classes, unwanted global accessibility, and inconvenience for unit test.

Overall, the detected use scenarios are usually associated with the modern software frameworks or programming techniques. This characteristic makes these use scenarios have a better practical value for developers than those mentioned in classical design pattern books.

3.4 Summary

There are 195 design pattern use scenarios detected from the Stack Overflow posts. These use scenarios could be reference cases for novice developers or examples of "known uses" when creating design pattern relevant documents.

4 Case Study Part 2: Related Design Pattern Pairs

In this section, we aim to investigate the following research question.

*RQ*2. Which pairs of design patterns may be related?

4.1 Motivation

Two design patterns may be taken as tags simultaneously in a same post. That means they are likely to be related in some way. In the GoF book, the "related patterns" attribute is described as "What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?" It implies two types of relatedness, i.e., the two design patterns are similar and the two design patterns are usually used together. The co-occurrence of design pattern tags could cover both of the two types. In this part, we aim to detect the related design pattern pairs from the co-occurrences. The results could help developers to better acknowledge the relationship between design patterns.

4.2 Approach

Firstly, the potentially related design pattern pairs were selected. For each given design pattern, we found out all the other design patterns co-occurring with, and sorted them in descending order according to the numbers of posts that contain the co-occurring pairs. By leveraging the knee point finding method [18], the design patterns ahead of the knee point were selected to form potential related pairs with the given design pattern. All the potential related pairs were determined in this way with regard to all the design patterns and the duplicates were eliminated, i.e., "PatternA-PatternB" and "PatternB-PatternA" counted as a same pair.

Next, each potential related design pattern pair was analyzed manually by consulting the involving topics. For each potential related pair, we clarified the topic assignments on the posts that contain the pair. The top most topic, which has the maximum times of assignments, was used to assist us to comprehend why the two design patterns co-occur. We extracted 15 keywords of the top most topic and 10 representative posts that are assigned to this topic and contain the pair as references.

Just as in the previous part, the manual analysis was conducted by the three participants. With these materials, each participant attempted to ascertain whether the pair of design patterns are indeed related in some way, and summarized why they are related. To help to understand the co-occurrence further, the participants could search for more materials that contain the two design patterns on the Internet. Afterwards, we discussed the results of each participant and tried to reach an agreement.

4.3 Results

By reviewing the top most topic of each selected co-occurring design pattern pair, we determined the related pairs. From some co-occurring pairs, no explicit relationships were inferred by our review, e.g., the two design patterns are merely building blocks of a same software system but do not interact with each other directly. Under this circumstance, the co-occurring design patterns were not regarded as related design patterns. Finally, we detected 70 unique pairs of related design patterns. Part of the related design pattern pairs are presented in Table 3 and the entire results can be accessed online[⑨]. For each related design pattern pair, we also provided a summary, which implies why they are related, and the online materials[⑨], which imply the relationship between the two design patterns.

Overall, the 70 related design pattern pairs involve 45 design patterns. Among them, the top 3 most frequent design patterns are "Dependency Injection", "Repository", and "MVC", which appear in 9, 9, and

⑨https://github.com/WoodenHeadoo/design-pattern-attributes/blob/master/supplements.pdf, July 2020.

Table 3. Related Design Pattern Pairs

Design Pattern Pair	Category	Summary	Online Materials[10]
Abstract factory & dependency injection	Co-operation	Abstract factory can be used in dependency injection frameworks for creating stateful objects	r1
Active record & MVC	Dependency	Active record can be the Model in MVC	r2
Active record & single table inheritance	Co-operation	Active record usually allows single table inheritance	r3
Command & MVVM	Dependency	Command is often used in MVVM architecture to coordinate the view with the viewmodel	r4
CQRS & event Sourcing	Co-operation	CQRS is often used along with event sourcing for efficient queries	r5
DAO & DTO	Analogy	They all operate on data between classes or modules	r6
DAO & record set	Dependency	Record set is often used in DAO to manipulate data	r7
DAO & repository	Analogy	They are all responsible for data access of a software system	r8
Dependency injection & factory	Analogy	They all have the purpose to separate the use of a certain component	r9
Dependency injection & reflection	Dependency	Dependency injection can be implemented by using reflection	r10
Dependency injection & repository	Co-operation	Repositories can be injected via dependency injection	r11
Dependency injection & singleton	Analogy	They can all make dependencies for objects	r12
Double Checked locking & singleton	Dependency	Double checked locking can be used to make singleton thread safe	r13
HMVC & MVC	Variation	HMVC is a variation of MVC	r14
Iterator & visitor	Analogy	Both iterator and visitor can be used to visit structures of elements	r15
Master/slave & sharding	Analogy	They can be all database partitioning approaches	r16
Mediator & publish/subscribe	Dependency	Mediator can be used to implement the publish/subscribe model	r17
Message broker & messaging	Variation	Message broker is a way for messaging	r18
Messaging & publish/subscribe	Variation	Publish/subscribe is a kind of messaging pattern	r19
MVC & MVVM	Analogy	MVC and MVVM are both for building architectures of presentation	r20
MVC & observer	Dependency	Observer can be used to synchronize the Model and the View in MVC	r21
MVC & service layers	Co-operation	Service layers interact with the controller in MVC	r22
Repository & unit of work	Co-operation	Unit of work is often implemented on repositories	r23

8 pairs, respectively. As presented in Subsection 3.3, these design patterns all have relatively large numbers of use scenarios. It shows again that they are active design patterns. In addition, the most frequently reviewed topic is topic #51, which is about examples and comparisons of design patterns. It represents the type of relatedness that the two design patterns of the pair are similar.

By analyzing each of the related design pattern pairs, we find that they may be divided into four categories.

• *Analogy.* The two sides of the pair are distinct design patterns but they have similar functions for some special uses. This category is relevant to the topic of comparison or difference between design patterns. The askers in Stack Overflow may post questions like: "what is the difference between A and B?" "A or B, which one to use?" For example, "DAO" and "Repository" are similar as they are all responsible for data access of a software system. "DAO" is closer to the database but "Repository" is associated with the domain model (r8). Likewise, "Dependency Injection" and "Singleton" are both design patterns that can provide an object with references to other objects. "Dependency Injection"

[10] https://github.com/WoodenHeadoo/design-pattern-attributes/blob/master/supplements.pdf, July 2020.

achieves this by explicitly setting the dependencies but "Singleton" makes the dependent objects globally accessible (r12).

• *Variation.* One design pattern in the pair is a variation, evolution, implementation, or subtype of the other one. In Stack Overflow, if one design pattern is discussed in a question post, the other one is also likely to be mentioned as a relevant term. For instance, "HMVC" is an evolution of the "MVC" design pattern to handle the salability problem of MVC-based applications (r14). "Publish/Subscribe" is a kind of "Messaging" design pattern that implements messaging between publishers and subscribers through an intermediary message bus (r19).

• *Co-Operation.* The two design patterns usually co-operate with each other to achieve some special purposes, but they are relatively independent. For example, "Repository" usually accompanies with "Unit of Work" to create an abstraction layer between business logics and data stores. "Repository" defines the basic operations of data entities and "Unit of Work" ensures multiple operations complete or fail entirely to avoid database in-consistency (r23). Another pair is "CQRS" and "Event Sourcing". "CQRS" is a design pattern that segregates the data reading model and the data writing model. Along with "Event Sourcing", the writing operations are stored as events that can be used to notify the reading model for more efficient queries (r5).

• *Dependency.* One design pattern in the pair is used by or assist to implement the other one. For example, "DAO" provides some specific data operations for the business logic layer without exposing details of the database. To perform the operations, "DAO" usually uses another design pattern, i.e., "Record Set", to manipulate the data by creating snapshots of the database (r7). In multi-threaded environments, we should ensure thread safety when using "Singleton" design pattern. That is, only one singleton instance can exist at the same time. To this end, "Double Checked Locking" is applied to the initialization method of the singleton class to check whether an instance is already created (r13).

Generally, the first two categories correspond to the type that the two design patterns are similar and the last two categories correspond to the type that they are usually used together. The numbers of pairs belonging to the four categories are 33, 6, 17, and 14, respectively. In design pattern books, the "related patterns" of one design pattern may only involve the other ones depicted in the current book. However, our results are not restricted by this factor. Design patterns of different collections can be related.

4.4 Summary

Via the case study, 70 related design pattern pairs were detected from Stack Overflow posts. These pairs could help developers to discriminate design patterns or serve as potential candidates for "related patterns" of design pattern relevant documents.

5 Discussions

5.1 Implications

The findings of this study would suggest some implications for software practitioners. On the one hand, developers, especially novice developers, could better acknowledge the usage of design patterns in practice. For example, the use scenarios provide some valuable instances to show when or where to apply a design pattern, especially along with the modern software frameworks or programming techniques. The related design pattern pairs could help developers to cognize and discriminate similar design patterns. On the other hand, these findings supply materials for documenting design patterns. For example, the use scenarios and the related design pattern pairs could serve as the contents of the sections of "known uses" and "related patterns" respectively in a design pattern book. As the online posts are updating continuously, these mined materials can keep up-to-date, too.

5.2 Automation

In this study, the final outcomes, i.e., the design pattern use scenarios and related design pattern pairs, were obtained manually. As the Q&A websites, e.g., Stack Overflow, are keeping changing, it will be much more convenient and efficient to update the results if we can automate the whole process. However, it is not a trivial job. For example, recognizing a use scenario from text and creating a human-readable depiction are both related to human cognition, and the answers tend to be open-ended. Nevertheless, we may go a step further. For instance, the automatic methods could be leveraged to summarize the topics, or to extract the keywords to describe the use scenarios or the relationships between design patterns, so that the labor cost could be reduced. We will consider these trials in the future work.

5.3 Threats to Validity

There are several potential threats to the validity of our work. First, the method of extracting the Stack Overflow posts relevant to each design pattern may involve inaccuracies. However, the method that uses the tag information is a practical way to identify domain specific posts [9, 15, 19]. The manual check of the tags also helps to reduce the inaccuracies. Second, the parameter settings of the topic model may be not the optima. To minimize this threat, we have tried to specify these settings objectively by combining the default values and the results of experiments. Third, the meanings of the topics were interpreted manually. It may lead to biases. To alleviate this problem, we referred to not only the keywords of high probabilities but also a set of example posts to comprehend the topics better. At last, when analyzing the use scenarios or related design patterns of a design pattern, we only investigated a part of the topics. We may obtain more outcomes if more topics are involved. However, we have found that most of the posts relevant to a design pattern can be covered by some certain topic. It is a reasonable choice to focus on some critical topics when considering the factor of efficiency.

6 Related Work

In this section, we briefly review the related studies to our work, including studies about the features of design patterns, empirical studies on Stack Overflow, and studies applying topic models to software documents.

6.1 Features of Design Pattern

There are several studies that investigate various features of design patterns with data. Similar to our work, some of them concern the features of a design pattern itself. Hasheminejad and Jalili extracted the problem domains of design patterns from design pattern collections and used them for design pattern selection [20]. Scanniello *et al.* investigated how the kind of documentation for design patterns affects the comprehensibility of design pattern instances in source code [21]. Ampatzoglou *et al.* examined the stability of GoF design patterns with a case study on about 65 000 Java classes [22].

Meanwhile, some studies focus on the features of software systems caused by applying design patterns. Hussain *et al.* performed a case study of the correlation between design pattern usage and system structural complexity [23]. Jaafar *et al.* evaluated the impact of six design patterns on the fault-proneness of three open-source software systems [24]. Aversano *et al.* explored the defects of design pattern classes in the crosscutting code by analyzing the software evolution data [25].

Different from these studies, the features concerned in this paper are use scenarios and design pattern relatedness.

6.2 Empirical Study on Stack Overflow

A number of empirical studies have leveraged the Stack Overflow as their data source. Among them, the studies attempting to summarize the themes, trends, or issues of some specific domains are most related to ours. Yang *et al.* conducted a large-scale study on Stack Overflow questions to explore security related topics and trends [15]. Similarly, the study conducted by Bagherzadeh and Khatchadourian concentrated on big data questions and tried to understand the interesting and difficult topics among big data developers [9]. Zou *et al.* performed an exploratory analysis of Stack Overflow posts aiming to comprehend the non-functional requirements of developers [26].

Some studies aim to discover the improper practice of programming from Stack Overflow. Nagy and Cleve investigated the MySQL related questions on Stack Overflow to analyze the error patterns in SQL queries [27]. In [28], Zhang *et al.* extracted API usage patterns from Java repositories and used them to discover potential API usage violations that may produce unexpected behaviors in Stack Overflow posts. Rahman *et al.* conducted a systematic analysis of insecure Python code blocks in Stack Overflow answers [29].

Furthermore, some other studies focus on the human or community behaviors on Stack Overflow. Ford *et al.* used mixed methods to identify contribution barriers for females on Stack Overflow [30]. Zhang *et al.* investigated the process that an answer on Stack Overflow becomes obsolete and identified the characteristics of obsolete answers [31]. Marder carried out a regression analysis of user activities on Stack Overflow to examine how the users behave when earning badges [32].

The key difference between our work and the above studies is that we focus on the analysis of design-pattern relevant question posts on Stack Overflow.

6.3 Topic Model in Software Engineering

Except for being applied to analyze the textual content of Stack Overflow, topic models have been lever-

aged in many other software engineering tasks. In some studies, topic model techniques are used for modeling source code. Thomas *et al.* leveraged a topic model technique on the source code of software systems to capture the changes and describe the evolutions [33]. To facilitate the reading and browsing of source code in code editors, Fowkes *et al.* used a scoped topic model on code tokens to fold code automatically so that the users can hide code blocks selectively [34]. Considering the barriers that prevent classical topic models from being applied to source code, e.g., the sparsity of source code, Mahmoud and Bradshaw proposed a specialized topic modeling approach for source code [35].

Moreover, some studies use topic models on other software documents. Xia *et al.* proposed a novel topic modeling algorithm for bug reports and used it for bug triaging [36]. In [37], Jiang *et al.* built a framework of recommending relevant API tutorial fragments, in which the topic model is used to estimate the correlations between tutorial fragments and APIs. Hu *et al.* analyzed the commit messages of software projects with an online topic model to discover emerging topics in development iterations [38].

Among these studies, many of them choose LDA as the topic model. In contrast, our choice is the specialized GSDMM, which is more suitable for our circumstance. On the one hand, the topic distribution for each design pattern can be obtained so that we can acknowledge what the important topics for a given design pattern are. On the other hand, GSDMM assigns each post only one topic instead of a mixture of topics; therefore we can better understand a topic by consulting the corresponding posts.

7 Conclusions

In this work, we conducted a case study on Stack Overflow to explore some specific attributes of design patterns, i.e., use scenarios and related design pattern pairs. We first extracted the Stack Overflow question posts relevant to design patterns by leveraging the tags, and then we used an advanced topic modeling technique to find the topics from the posts and build the associations between the topics and each design pattern. With the help of the identified topics, we detected 195 use scenarios of 61 design patterns and 70 pairs of related design patterns. These use scenarios are in relation to various software frameworks and techniques; the related pairs mainly belong to four categories, namely analogy, variation, co-operation, and dependency.

These findings, on the one hand, could help developers better acknowledge the usage and relatedness of design patterns in modern programming practice. On the other hand, they could serve as materials for creating design pattern relevant documents, e.g., as the examples of "known uses" and "related patterns" in design pattern books.

In the future, we plan to investigate other online resources, such as other Q&A websites, forums, and technical blogs, to detect more use scenarios and related design pattern pairs. We also plan to extend this work to make the methodology more automatic.

References

[1] Zhang C, Budgen D. What do we know about the effectiveness of software design patterns? *IEEE Trans. Softw. Eng.*, 2012, 38(5): 1213-1231.

[2] Henninger S, Corrêa V. Software pattern communities: Current practices and challenges. In *Proc. the 14th Conf. Patt. Lang. Prog.*, Sept. 2007, Article No. 14.

[3] Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software (1st edition). Addison-Wesley Professional, 1994.

[4] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. Pattern-Oriented Software Architecture: A System of Patterns (1st edition). Wiley, 1996.

[5] Barua A, Thomas S W, Hassan A E. What are developers talking about? An analysis of topics and trends in Stack Overflow. *Empir. Softw. Eng.*, 2014, 19(3): 619-654.

[6] Ahmad A, Feng C, Ge S, Yousif A. A survey on mining stack overflow: Question and answering (Q&A) community. *Data Technol. Appl.*, 2018, 52(2): 190-247.

[7] Hindle A, Godfrey M W, Holt R C. What's hot and what's not: Windowed developer topic analysis. In *Proc. the 25th IEEE Int. Conf. Softw. Maint.*, Sept. 2009, pp.339-348.

[8] Han D, Zhang C, Fan X, Hindle A, Wong K, Stroulia E. Understanding Android fragmentation with topic analysis of vendor-specific bugs. In *Proc. the 19th Working Conf. Reverse Eng.*, Oct. 2012, pp.83-92.

[9] Bagherzadeh M, Khatchadourian R. Going big: A large-scale study on what big data developers ask. In *Proc. the 27th ACM Joint Eur. Softw. Eng. Conf./Symp. Found. Softw. Eng.*, Aug. 2019, pp.432-442.

[10] Zhou P, Liu J, Yang Z, Zhou G. Scalable tag recommendation for software information sites. In *Proc. the 24th IEEE Int. Conf. Softw. Anal. Evol. Reeng.*, Feb. 2017, pp.272-282.

[11] Chen C, Gao S, Xing Z. Mining analogical libraries in Q&A discussions — Incorporating relational and categorical knowledge into word embedding. In *Proc. the 23rd IEEE Int. Conf. Softw. Anal. Evol. Reeng.*, Mar. 2016, pp.338-348.

[12] Wang X Y, Xia X, Lo D. TagCombine: Recommending tags to contents in software information sites. *J. Comput. Sci. Technol.*, 2015, 30(5): 1017-1035.

[13] Zhang Y, Lo D, Xia X, Sun J L. Multi-factor duplicate question detection in Stack Overflow. *J. Comput. Sci. Technol.*, 2015, 30(5): 981-997.

[14] Jiang H, Liu D, Chen X, Liu H, Mei H. How are design patterns concerned by developers? In *Proc. the 41st Int. Conf. Softw. Eng. Comp.*, May 2019, pp.232-233.

[15] Yang X L, Lo D, Xia X, Wan Z Y, Sun J L. What security questions do developers ask? A large-scale study of Stack Overflow posts. *J. Comput. Sci. Technol.*, 2016, 31(5): 910-924.

[16] Yin J, Wang J. A Dirichlet multinomial mixture model-based approach for short text clustering. In *Proc. the 20th ACM Int. Conf. Knowl. Disc. Data Mining*, Aug. 2014, pp.233-242.

[17] Griffiths T L, Steyvers M. Finding scientific topics. *Proc. National Academy Sci.*, 2004, 101(suppl. 1): 5228-5235.

[18] Satopaa V, Albrecht J, Irwin D, Raghavan B. Finding a "Kneedle" in a haystack: Detecting knee points in system behavior. In *Proc. the 31st Int. Conf. Distrt. Comput. Syst. Workshops*, June 2011, pp.166-171.

[19] Rosen C, Shihab E. What are mobile developers asking about? A large scale study using Stack Overflow. *Empir. Softw. Eng.*, 2016, 21: 1192-1223.

[20] Hasheminejad S M H, Jalili S. Design patterns selection: An automatic two-phase method. *J. Syst. Softw.*, 2012, 85(2): 408-424.

[21] Scanniello G, Gravino C, Risi M, Tortora G, Dodero G. Documenting design-pattern instances: A family of experiments on source-code comprehensibility. *ACM Trans. Softw. Eng. Methodol.*, 2015, 24(3): Article No. 14.

[22] Ampatzoglou A, Chatzigeorgiou A, Charalampidou S, Avgeriou P. The effect of GoF design patterns on stability: A case study. *IEEE Trans. Softw. Eng.*, 2015, 41(8): 781-802.

[23] Hussain S, Keung J, Khan A A, Bennin K E. Correlation between the frequent use of gang-of-four design patterns and structural complexity. In *Proc. the 24th Asia-Pacific Softw. Eng. Conf.*, Dec. 2017, pp.189-198.

[24] Jaafar F, Guéhéneuc Y G, Hamel S, Khomh F, Zulkernine M. Evaluating the impact of design pattern and anti-pattern dependencies on changes and faults. *Empir. Softw. Eng.*, 2016, 21(3): 896-931.

[25] Aversano L, Cerulo L, Di Penta M. Relationship between design patterns defects and crosscutting concern scattering degree: An empirical study. *IET Softw.*, 2009, 3(5): 395-409.

[26] Zou J, Xu L, Yang M, Zhang X, Yang D. Towards comprehending the non-functional requirements through developers' eyes: An exploration of Stack Overflow using topic analysis. *Inf. Softw. Technol.*, 2017, 84: 19-32.

[27] Nagy C, Cleve A. Mining Stack Overflow for discovering error patterns in SQL queries. In *Proc. the 7th IEEE Int. Conf. Softw. Maint. Evolut.*, Sept. 2015, pp.516-520.

[28] Zhang T, Upadhyaya G, Reinhardt A, Rajan H, Kim M. Are code examples on an online Q&A forum reliable? A study of API misuse on Stack Overflow. In *Proc. the 40th Int. Conf. Softw. Eng.*, May 2018, pp.886-896.

[29] Rahman A, Farhana E, Imtiaz N. Snakes in paradise? Insecure python-related coding practices in Stack Overflow. In *Proc. the 16th IEEE/ACM Int. Conf. Mining Softw. Repos.*, May 2019, pp.200-204.

[30] Ford D, Smith J, Guo P J, Parnin C. Paradise unplugged: Identifying barriers for female participation on Stack Overflow. In *Proc. the 24th ACM Joint Eur. Softw. Eng. Conf./Symp. Found. Softw. Eng.*, Nov. 2016, pp.846-857.

[31] Zhang H, Wang S, Chen T P, Zou Y, Hassan A E. An empirical study of obsolete answers on Stack Overflow. *IEEE Trans. Softw. Eng.*. doi:10.1109/TSE.2019.2906315.

[32] Marder A. Stack Overflow badges and user behavior: An econometric approach. In *Proc. the 12th IEEE/ACM Int. Conf. Mining Softw. Repos.*, May 2015, pp.450-453.

[33] Thomas S W, Adams B, Hassan A E, Blostein D. Validating the use of topic models for software evolution. In *Proc. the 10th IEEE Working Conf. Source Code Anal. Manip.*, Sept. 2010, pp.55-64.

[34] Fowkes J, Chanthirasegaran P, Ranca R, Allamanis M, Lapata M, Sutton C. Autofolding for source code summarization. *IEEE Trans. Softw. Eng.*, 2017, 43(12): 1095-1109.

[35] Mahmoud A, Bradshaw G. Semantic topic models for source code analysis. *Empir. Softw. Eng.*, 2017, 22(4): 1965-2000.

[36] Xia X, Lo D, Ding Y, Al-Kofahi J M, Nguyen T N, Wang X. Improving automated bug triaging with specialized topic model. *IEEE Trans. Softw. Eng.*, 2017, 43(3): 272-297.

[37] Jiang H, Zhang J, Ren Z, Zhang T. An unsupervised approach for discovering relevant tutorial fragments for APIs. In *Proc. the 39th Int. Conf. Softw. Eng.*, May 2017, pp.38-48.

[38] Hu J, Sun X, Li B. Explore the evolution of development topics via on-line LDA. In *Proc. the 22nd IEEE Int. Conf. Softw. Anal. Evol. Reeng.*, Mar. 2015, pp.555-559.

Dong Liu received his M.S. degree in computer science and technology from Hebei University of Technology, Baoding, in 2016. He is currently a Ph.D. candidate in Dalian University of Technology, Dalian. His current research interests include mining software repositories and data-driven methods in software engineering.

Zhi-Lei Ren received his B.S. degree in software engineering and his Ph.D. degree in computational mathematics from Dalian University of Technology, Dalian, in 2007 and 2013, respectively. He is currently an associate professor with Dalian University of Technology, Dalian. His current research interests include evolutionary computation, automatic algorithm configuration, and mining software repositories.

Zhong-Tian Long is currently a undergraduate student in Dalian University of Technology, Dalian. His current research interests include machine learning and artificial intelligence.

Guo-Jun Gao received his B.S. degree in software engineering from Dalian University of Technology, Dalian, in 2010. He is currently a Ph.D. candidate in Dalian University of Technology, Dalian. His current research interests include search-based software engineering and compiler options selection.

He Jiang received his Ph.D. degree in computer science from University of Science and Technology of China, Hefei, in 2004. He is currently a professor with Dalian University of Technology, Dalian, and an adjunct professor with Beijing Institute of Technology, Beijing. His current research interests include intelligent software engineering, mining software repositories, and compilers.