Empirical Software Engineering (2022) 27: 8
https://doi.org/10.1007/s10664-021-10034-0

Mining and relating design contexts and design patterns from Stack Overflow

Laksri Wijerathna[1] · Aldeida Aleti[1] · Tingting Bi[1] · Antony Tang[2,3]

Accepted: 4 August 2021 / Published online: 23 October 2021

Abstract

Design contexts are factors that shape a design, and whilst they are recognised by developers, they are typically tacit. Unlike software requirements, software engineering researchers have paid little attention to design contexts and there is little or no systematic research on how design contexts influence design. In this paper, we conduct an empirical investigation using Stack Overflow with the aim of mining design context knowledge that is related to design patterns. We chose to study design patterns because they are clear and identifiable. In this work, we develop a new taxonomy of design context terms related to design patterns. We introduce a new automated mining approach, DPC Miner, for mining design context knowledge from Stack Overflow. Finally, we analyse the Stack Overflow posts and present how design context impacts decisions about design patterns in practice.

Keywords Design context · Design pattern · Collective knowledge mining · Stack overflow · Empirical analysis

Communicated by: Shaowei Wang, Tse-Hsun (Peter) Chen, Sebastian Baltes, Ivano Malavolta, Christoph Treude, and Alexander Serebrenik

This article belongs to the Topical Collection: *Collective Knowledge in Software Engineering*

✉ Laksri Wijerathna
laksri.wijerathna@monash.edu

Aldeida Aleti
aldeida.aleti@monash.edu

Tingting Bi
Tingting.Bi@monash.edu

Antony Tang
atang@swin.edu.au

1 Faculty of Information Technology, Monash University, Melbourne, Australia

2 Vrije Universiteit, Amsterdam, The Netherlands

3 Swinburne University of Technology, Melbourne, Australia

1 Introduction

Software design decisions, and in particular the decisions regarding design patterns can be made under different circumstances. These circumstances are known as design contexts. Design Context (DC) can be defined as "a representation of the external relevant information describing a situation that decision-makers are interested in to have access to, before a decision or during a decision process" (Carlson et al. 2016). The most relevant information is often not explicitly documented, and is in the form of implicit requirements and environmental factors (Tang et al. 2008). Although design context has a crucial role in shaping design decisions, our understanding of it is very limited. This is because, until now, there are very few studies on design contexts, and those that exist rely on case studies, interviews, and workgroups as research methods, which are limited in scope and can only cover a small range of contextual factors. With the very large number of contextual factors and combinations, (Dybå et al. 2012), the research to date has only scratched the surface of this very important area. Therefore, to cover the large combination of design context and related knowledge, a more diversified and large knowledge source would be beneficial to both knowledge expansion and future research.

Design patterns (DPs) are solutions to reoccurring problems in software design that have been proven to work well, and are often used by software developers (Washizaki et al. 2020; Riehle 2011). In this paper, we use the terminology DP to refer to a broader category of patterns that provide reusable solutions in the software design phase. As an example, a simple object-oriented pattern like Singleton and a general scope architecture pattern like Broker are both referred to as DP. There are many design patterns that programmers and developers have developed over the years. These design patterns serve different purposes such as providing better program structures, division of responsibilities in the structure of a system, or regulating communications between threads and processes. We broadly group them into three categories: programming, communication and architecture-related design patterns. Programming design patterns deal with object creation, such as abstract factory and builder, object-oriented structures, such as adaptor and bridge, and software behaviours, such as iterator and observer. Design patterns that operate at the architecture level are concerned with the structure of the software and how the different architectural elements are grouped together. Examples of such patterns are the client-server pattern and SOA. Finally, the communication-oriented design patterns deal with networking, communication and concurrency issues. Examples of such patterns are connection pooling and publish-subscribe pattern. While design patterns are well known and they are often accompanied by certain design guidelines, there is little knowledge on how design patterns are used in certain contexts. In discussion forums such as Stack Overflow, developers often asked if a certain design pattern can be used. The dialogue often comes to "it depends", in which developers clarify the environments of the application before deciding how to use the design pattern. The circumstances that underpin the applicability of a design pattern is its design contexts.

Applying a design pattern to solve a particular design problem requires some experience with knowing when and how to use design patterns. The effectiveness of a design pattern depends on many contextual factors, for which we do not have a comprehensive and structured knowledge repository that can help with this process. Instead, software developers often rely on community-based question-answer sites and online forums to discuss their design-related concerns. Stack Overflow is one of the most popular platforms where software developers have posted more than 21 million questions and 31 million answers so far, many of which are related to design patterns. This information repository is growing fast, and contains valuable knowledge about design patterns and their context, which is yet to be mined.

Previous studies have realised the great potential of Stack Overflow by successfully mining architecture-related knowledge. One such work is on the effective mining of architecture smells (Tian et al. 2019). The study uses a Grounded Theory method to analyse Stack Overflow posts and discover why architecture smells happen, the approaches and tools that developers use for detecting and refactoring architecture smells, the quality attributes affected, and difficulties in detecting and refactoring architecture smells. Soliman et al. (2016) extracts Stack Overflow knowledge related to architecture decisions by classifying the types of architecture-relevant posts in Stack Overflow. The study presents an analysis of the terms in Stack Overflow posts that distinguish architecture-relevant posts from programming posts. For a comprehensive overview of approaches that mine Stack Overflow we refer the reader to the survey by Ahmad et al. (2018).

The closest work to our study is the preliminary analysis of how developers discuss architecture patterns, quality attributes and design context in Stack Overflow (Bi et al. 2018). The study reports that the most frequently asked design question is of the type "should I use this architecture pattern in this application?", which indicates that context is a notable consideration in Stack Overflow posts. The authors argue that "developers often need to know certain information when designing with architecture patterns, such as the relationships between quality attributes and architecture patterns, characteristics and potential issues of using a pattern". These findings inspired our study, which is focused on discovering new design knowledge from empirical data that is regarded as important by developers when considering design patterns, with particular design contexts.

In this paper, we perform an empirical analysis of Stack Overflow posts to discover the contextual factors that influence the choice of design patterns. We consider design patterns at all three levels: programming, communication, and architecture. To deal with a large amount of information in Stack Overflow, the majority of which may not contain design context-related knowledge, we develop an automated mining approach that extracts design context-related posts. Since the literature has no consolidated taxonomy for design context required for labelling and analysing the posts, we develop a new DPC (Design Pattern Context) taxonomy. We employ a statistical stopping criterion to ensure that the taxonomy covers all relevant terms within a confidence interval. Finally, we label and analyse 1,500 Stack Overflow posts and discover how design context is considered in practice when deciding on a design pattern. In summary, this paper makes the following contributions:

– We introduce a consolidated taxonomy that covers context terms that are relevant to design patterns.
– We develop an automated mining approach that can successfully mine design context knowledge from Stack Overflow. The mining approach has an accuracy of 0.871, precision of 0.815, recall of 0.88, and Area Under the Receiver Operating Characteristic (ROC) Curve of 0.87.
– We present an analysis of how design contexts is discussed in relation to the design pattern decisions in practice.

2 Background

2.1 Design Patterns

A software design pattern is a solution to solving a problem that occurs over and over again (Gamma et al. 1995). Since its inception in the early 90's, as documented in (Gamma et al. 1995; Buschmann and Henney 1993), the general concept of design pattern (DP) has been used in software design at different levels of design abstraction. At the programming level, patterns have been used to address object creation (e.g. abstract factory and builder), object-oriented structures (e.g. adaptor, bridge, MVC), software behaviour (e.g. Iterator, Observer). There are also design patterns that address software structure design. That is, they are solutions to structure software by their architectural elements (Bass et al. 2012). Examples of design patterns include layered pattern, client-server pattern, multi-tier pattern, and SOA. There are also design patterns that address networking, communication and concurrency issues (Schmidt et al. 2013). Examples are connection pooling and publish-subscribe patterns. Different design patterns have been created for specific design issues at different levels of design abstraction such as architecture design, programming design, and communication design.

2.2 Design Contexts

In addition to requirements, many environmental factors influence software design but they are not so well defined. We call these factors design contexts. Software design is very much context dependent. Consider the difference between a bank and a travel agency. The design contexts can dictate what kind of system each business would end up with and that could be vastly different. A bank would have to satisfy design contexts such as financial institution regulations, IT standards, and security guidelines. These contexts could guide an architect to end up designing a multi-tier SOA architecture. Whereas a travel agency may have limited budget and IT expertise that would likely drive it to acquire a software package. Even though design contexts are forces that influence design decisions, they are not stated explicitly and are often tacit.

Design context (DC) is an intuitive concept and it is very hard to define due to its highly dynamic nature (Chattopadhyay et al. 2018). Design contexts can be seen as “conditions that influence design decisions but they are not specified explicitly as requirements” (Tang et al. 2008). Harper and Zheng (2015) suggest that design contexts are forces (internal and external) that influence stakeholders’ concerns . Dybå et al. (2012) argue that technology selection can be influenced by different contexts or environmental factors. Power and Wirfs-Brock found that sometimes design decisions are made in a narrow context of feature delivery, ignoring long-term issues that may affect a design. They also found that geographic distributions of software developers and trust relationships are factors that influence architecture decisions (Power and Wirfs-Brock 2018). Kitchenham et al. (2002) suggested to ”specify as much of the industrial context as possible”.

There are some works that categorise design contexts. Bedjeti et al. (2017) identified four context categories of the viewpoint (i.e., platform context, user context, application context, and organizational context) . Petersen and Wohlin (2009) provided a checklist for documenting design contexts from six perspectives: product, processes, practices and techniques, people, organization, and market . Carlson et al. developed a context model through a workshop for architectural decision making. The main categories of the context model are organization, product, stakeholder, development method and technology, market and business (Carlson et al. 2016). Groher and Weinreich studied environmental factors that influence decision making. They suggested eight categories: Company Size, Business Factors, Organizational Factors, Technical Factors, Cultural Factors, Individual Factors, Project Factors, and Decision Scope (Groher and Weinreich 2015). Bi et al. extracted posts from Stack Overflow to investigate design contexts, which were categorized into application context, platform context and organizational context (Bi et al. 2018).

Dybå et al. found interrelated contexts such as geographic location, human factors, cultures that influence technology decisions (Dybå et al. 2012). They classify contexts into omnibus contexts and discrete contexts. Omnibus contexts specify the phenomenon (what), subjects (who), location (where), time (when), and rationale (why). Discrete contexts have three classes: technical, social, and environmental. They further argue that there are ”infinite number of contextual factors and combinations to consider” and as such ”do not expect a single, precise, technical definition of context in SE”.

Although design contexts are important in shaping design decisions, there are only limited empirical studies on what they are and even fewer studies on how they shape design decisions. The empirical works that are cited mostly rely on case studies, interviews, and workgroups as research methods. With the ”infinite number of contextual factors and combinations” (Dybå et al. 2012), research in this area is scarcely enough. We explore design contexts in relation to design patterns for a number of reasons. First, there is a large amount of empirical data in Stack Overflow that can tell us what design contexts are considered by developers when they are dealing with design patterns. Second, design patterns are well-defined concepts in the software community and it is an atomic entity that is easier to relate with design contexts than less well-defined entities such as an application system. Third, we can use machine learning methods to mine large amount of data from Stack Overflow to produce attainable results that relate these two subjects.

2.3 Mining Architecture Information from Discussion Forums

There are several work done in architecture-related information extraction from discussion forums. Velasco-Elizondo et al. applied information extraction techniques (i.e., entities extraction) and knowledge representation (i.e., ontology) based approach to automatically analyse architecture patterns in terms of quality attributes (e.g., Performance) (Velasco-Elizondo et al. 2016). Specifically, the approach uses an ontology that contains two sub-types of ontologies. One is English grammar-based ontology, and the other is performance ontology that defines performance-specific concepts (e.g., throughput). To identify the relationships between architecture patterns and quality attributes, from architecture pattern descriptions or specifications, they used information extraction techniques (i.e., entity extraction) and an ontology. The experiment results show that the approach helps inexperienced architects select architecture patterns by knowing whether specific quality attributes are promoted or inhibited.

Mirakhorli et al. compared six classification algorithms (SVM, C45, Bagging, SLIPPER, Bayesian logistic regression, AdaBoost) to semi-automatically identify architecture tactics from source code (Mirakhorli and Cleland-Huang 2016). In another work, (Mirakhorli et al. 2012), Mirakhorli et al. applied classification techniques and information retrieval to identify architecture tactic-related classes in source code. This approach can be used to construct traceability links between source code and architectural tactics automatically. In addition, their approach helps minimise the human efforts for building traceability that can be used to support maintenance activities and prevent architectural erosion.

Casamayor et al. (2012) proposed an approach that applies NLP techniques and the K-means algorithm to categorise candidate responsibilities into groups. The input of this approach is requirement documents, and then the approach applied the POS tagging technique to detect the actions and tasks that the system needs. The authors then applied K-means to group similar responsibilities into architectural components. The results show that grouped architectural components by using this approach correspond to the expected ones made by experts. Gokyer et al. (2008) applied NLP and machine learning techniques in requirements documents to classify related NFRs into architectural concerns. This approach can guide architects in linking architectural concerns from the problem domain to the running components and connectors of the solution domain. Zhang et al. (2006) utilised NLP techniques (e.g., POS tagging) and ontology-based analysis in architecture documents and source code to identify potential components. This approach can mine architecture knowledge (e.g., data and control communications between components and implemented design patterns), which can assist the maintainers to comprehend the system architecture by identifying components and studying their properties.

2.4 Machine Learning Approaches for Mining Software Engineering Knowledge

There are several machine learning based approaches adopted in mining software engineering (SE) related knowledge from various data sources, such as software repositories and discussion forums. For information retrieval, various Natural Language Processing (NLP) based Language Models (LM) are used. These LMs can predict the probability of words by analysing the sequence of linguistic units such as words, sentences, or documents in data. Primarily, there are two types of language models: 1) Statistical language models and 2) Neural Language Model. Statistical Language Models estimate a probability distribution while Neural Language Model learns a distributed representation for given tokens(i.e., words, sentences) of input data.

Statistical language models (SLM) are capable of determining the probabilistic sequence of words (Goodman 2001) or a given tokens of language depending on the application. These language models can be in the form of uni-gram, n-gram, bidirectional, exponential, and adaptive models (Rosenfeld 2000). Uni-gram is the simplest form of the language model and evaluates each word independently without considering the context in the probabilistic calculation. The n-gram approach can create a probability distribution for a sequence of n, where n can be any number (Song and Croft 1999). The goal of a trained language model is to assign a low probability to less meaningful words while assigning a higher probability to more human-friendly words, after learning the joint probability functions from sequences of words. Variations of SLM approaches are used in SE-related knowledge mining applications, such as making programming language code suggestions (Hindle et al. 2016), defect code detection (Choi et al. 2011), and source code repository mining (Allamanis and Sutton 2013b). However, these SLMs suffer from the “curse of dimensionality” due to the massive scale of possible sequences of words in a language (Bengio et al. 2003).

Neural Language Models (NLM) are designed to overcome the “curse of dimensionality” problem by using a distributed representation of words (Bengio et al. 2003). These models learn the distributional representation of words and represent the words as a non-linear combination of weights, thus achieves the dimensionality reduction. Therefore, NLMs are more computationally feasible in larger datasets compared to SLM (Mikolov et al. 2013a). NLMs are used in many software engineering related knowledge mining approaches, such as predicting the programming language of questions from Stack Overflow (Alreshedy et al. 2018), summarise answers for developers questions (Xu et al. 2017), semantically related question identification (Xu et al. 2016; Zhang et al. 2017), design pattern prediction (Liu et al. 2020).

Further, there are non-statistical information retrieval approaches like Vector Space Model (VSM) and latent Semantic Indexing (LSI) (Thomas 2011). Initially, VSM used for automatic document indexing (Salton et al. 1975) and later expanded to text classification and text mining. VSM creates a unique set of words given a document and maps each word into a numeric value. This numerical value is used as a weight to indicate the overall impact of the particular word in training. VSM uses different weighting techniques such as Term Frequency and Term Document Frequency (Ali et al. 2018). VSM-based approaches are used in classifying GitHub repositories (Cai et al. 2016) and for software design pattern classification (Hussain et al. 2017). LSI also uses the underlying latent semantic structure of the data for document indexing as VSM. LSI generates a co-occurrence term matrix but uses a Singular Value Decomposition(SVD) to reduce the co-occurrence matrix and eliminate the noise in the document by retaining the latent semantics of the document (Deerwester et al. 1990). LSI learns latent topics by matrix decomposition. LSI-based approaches are used in concept extraction from source code (Marcus et al. 2004; Marcus et al. 2005; Poshyvanyk et al. 2006), understanding conceptual relationships between source code entities (Chen et al. 2012), and software evolution change impact analysis (Borg et al. 2017).

Latent Dirichlet Allocation (LDA) is a generative probabilistic model that uses the hierarchical Bayesian model (Blei et al. 2003). Given a set of documents, LDA uses the word distribution in the documents and represents the documents as a random mixture of latent topics. LDA is often used in tasks like classification, document summarising, and similar document identification. The LDA-based approaches are used in software artifacts mining, such as automatically categorising software systems in open source repositories (Tian et al. 2009) and archives (Kawaguchi et al. 2003). Further, in mining software concepts from source code (Linstead et al. 2007a; Linstead et al. 2007b; Lukins et al. 2008), explaining software defects (Chen et al. 2012), finding main discussion topics in developer discussions (Barua et al. 2014) and mining source code repositories (Allamanis and Sutton 2013a). The drawback of LDA is that it does not work well with short texts (Zhang et al. 2017).

While several machine learning models are used for SE-related tasks, there are no automated mining approaches aimed at extracting design context knowledge. To mine domain-specific knowledge such as design context from a crowd-sourced repository like Stack Overflow, we require a machine learning approach that is effective with short text. Therefore, in this paper, we implement a continuous skip-gram model-based Representation Learning (RL) model, enhanced with collocations to learns high-quality semantic and syntactic word embedding (Mikolov et al. 2013a) for design pattern context.

3 Motivation and Research Questions

As discussed in Section 2.2, software design is very much context dependent and design context plays an important part in shaping design decisions. As such, we conjecture that using design patterns is dependent on design contexts, except we do not know what design contexts influence which design pattern. While there is some existing work in identifying and modelling context elements of a software system (Bedjeti et al. 2017) and on understanding design context concerning design decisions, such as architecture patterns (Bi et al. 2018), there are no previous studies that provides a comprehensive taxonomy of design pattern contexts based on empirical data. The research questions we aim to answer in this paper are:

RQ1. What contextual factors that are relevant to practitioners when considering a design pattern? Usually, design context factors are not specified explicitly as requirements, thus they remain tacit. Further, due to the scarcity of empirical evidence and an infinite number of contextual factors and combinations, most of the contextual factors and their influence in design patterns are not sufficiently documented or communicated. Therefore, the contextual factors can be overlooked or ignored, and practitioners might miss essential knowledge that might be leading to a particular design pattern(s) selection. However, if such knowledge is classified and readily available, the aforementioned issue can be avoided. In this work, we explore such knowledge through an empirical analysis, and provide an overview of the relation of design context on design patterns.

Exploring and identifying design contextual knowledge from knowledge repository would be a nearly impossible task without a proper classification mechanism. We found no readily available taxonomy with respect to design context and design patterns. On the contrary, from several literature we found the traces of knowledge which could be formulated into an initial DPC taxonomy. We generated an initial taxonomy by analysing several literature sources. The use of this taxonomy is to get an overall structure in knowledge exploration. Along the process we refined DPC taxonomy, which allow us to make more reliable knowledge exploration with a reliable set of labels.

RQ2. How can we automatically mine design context knowledge related to design patterns? Existing research on design context mostly relies on case studies, interviews, and workgroups as research methods. While these existing studies helped us in getting initial insights about design context, these research methods are confined by the number of participants/cases that can be part of the study at a given time frame, hence it is possible that they may miss important contextual factors. The crowd-sourced Q&A sites like Stack Overflow provide a tremendous amount of information, which allows us to mine an extensive amount of knowledge. Software engineers frequently share the problems they encountered in designing, developing, and testing software solutions with eminent background information. Further, there are a lot of new post creations on a daily basis in the Stack Overflow hence, it provides a tremendous growing knowledge structure that the future researchers can use to update/expand the knowledge curation.

For an enormous crowd-sourced knowledge repository like Stack Overflow, the manual mining process would be a labour-intensive process and practically impossible on a large scale. On the contrary, to extract knowledge an automated and reliable mining method would have many merits. To address this concern, we develop a Design Pattern Context (DPC) Miner, which is an automated approach for mining and classifying posts that contain design context knowledge related to design patterns.

RQ3. What Design Context terms are discussed in relation to Design Patterns? Design patterns are used under different constraints and opportunities, which form the context of where a particular DP is or isn't effective. As an example, in a scenario where a developer is building a web application, the design pattern which is selected would be different depending on whether the application is an enterprise system or a mobile app. There is no comprehensive knowledge on what contexts are discussed with DPs. Our research goal is to gather this knowledge systematically from empirical data.

4 Methodology

Context is hard to study due to the methodological and theoretical problems that can arise due to study-to-study variations in the research finding (Dybå et al. 2005). Empirical research methods help identify universal relationships that are independent from work and process settings (Dybå et al. 2005). A possible way of doing empirical research is to "collect qualitative data that illuminate context effects and interactions ... that can aid in making inferences about the situation" (Dybå et al. 2012). Using an empirical method, we collect qualitative data from Stack Overflow to help us identify the relationship between design context and design patterns. An empirical methods allows us to "measure multiple dependent variables that can uncover situational context when used in conjunction with one another or explain the gap in meaning" (Dybå et al. 2012). We analyse the relationship between design context and design patterns, which helps us reflect on what context is, and what contexts are discussed in design decisions, and in particular, the choice of design patterns.

As shown in Fig. 1, Step 1 of our methodology is the set-up. In this step, we initially explore the DC and DP (explained in Section 4.1.1) to helps us to create an initial set of keywords, which forms the preliminary taxonomy. The initial exploration of DP and DC is used also to identify a set of design patterns that helps us filter the Stack Overflow posts (Section 4.1.2).

Step 2 is the Design Pattern Context (DPC) Taxonomy generation. From the list of keywords identified during Step 1.2, we create an initial keyword-based taxonomy (Step 2.1), described in Section 4.3.1. This taxonomy is created from keywords found in the initial exploration of DP and DC (Step 1.1), and may not be representative of the context terms discussed by software developers. Hence, we extract and label (Step 2.2) Stack Overflow posts to evolve (Step 2.3) and consolidate this initial taxonomy (Step 2.4). Data labelling happens in several stages (Step 2.2, 3.1, and 4.2), described in Section 4.2. Section 4.3.2

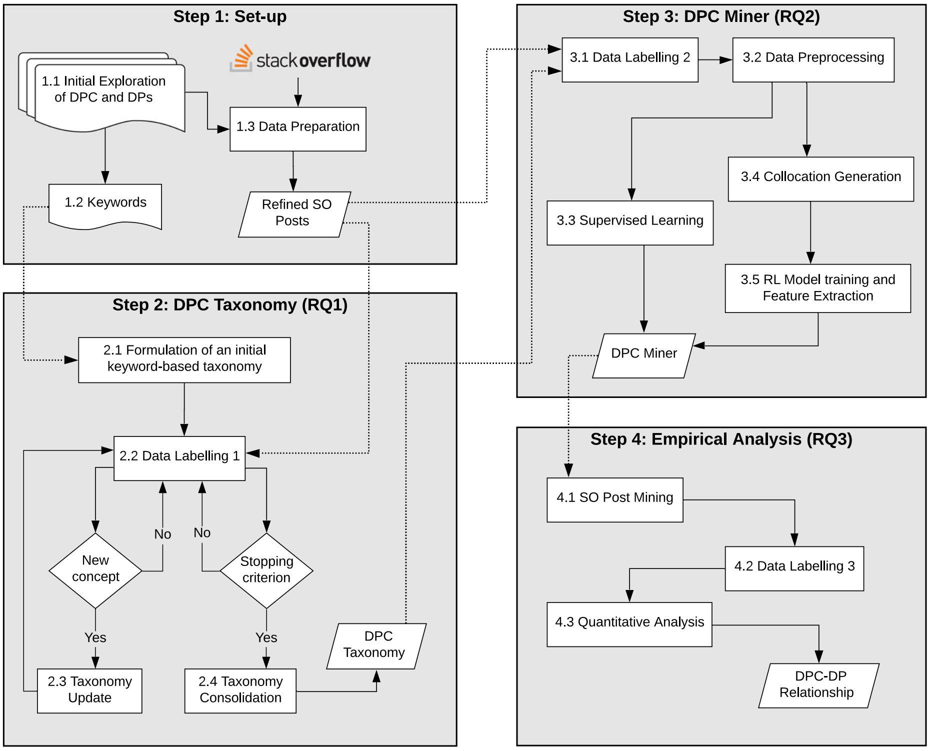


Fig. 1 Overview of the methodology

explains how we update the taxonomy, the stopping criterion for deciding when to consider the context terms complete, and the taxonomy consolidations step. The output of this step is a final taxonomy of design context terms considered in relation to design patterns, which answers RQ1.

Step 3 focuses on answering RQ2 and is aimed at developing an automated mining approach for extracting design context knowledge from Stack Overflow (described in detail in Section 4.4). First, the data is preprocessed (Step 3.2), as described in Section 4.4.1. Next, we create a domain specific feature learning model (Step 3.5), which learns the word embedding by capturing the semantic and syntactic similarity of the words from the DP question posts. To enhance the learning capability of the feature learning model, we first generate collocations (Step 3.4). In the textual content, some words hold more meaning when they are considered together as a collocation. As an example, the term "*web application*" has more meaning in context identification than when considered as separate words, "*web*" and "*application*". In the collocation generation, we automatically identify these collocations using the syntactic structure of the sentence and feed them as features to the Representation Model (RL). The generation of collocations is described in Section 4.4.2. We investigate eight different supervised learning algorithms (Step 3.3) which are described in Section 4.4.4. The output of Step 3 is DPC Miner which answers RQ2.

Step 4 of our methodology is focused on answering RQ3, which is about analysing the relationship between design context and design patterns. First, we employ the DPC Miner to mine more Stack Overflow posts (Step 4.1) that contain design pattern-related context terms (described in Section 4.5.1). These posts are manually labelled (Step 4.2) following the method described in Section 4.2 and quantitatively analysed (Step 4.3) as described in Section 4.5.2. Finally, the results of this quantitative analysis are presented in Section 5.3.

4.1 Set-up

The overview of the Set-up is shown in Fig. 1 - Step 1. We use the published literature and Stack Overflow as our input sources to derive a list of common terms that describe design context and prepare the data downloaded from SO.

4.1.1 Initial Exploration of DPC and DPs

To mine knowledge from empirical data, it is essential to have an overview of the knowledge factors that we are going explore. Therefore, to gain insight of the preliminary overview of DPC and DPs, first, we explore the published literature in two aspects: 1) literature on design patterns, and 2) literature studying design context. This step is important for the process of data preparation, as it provides a simple guideline that we can follow in the data extraction.

From the design pattern-related literature (Gamma et al. 1995; Bass et al. 2012; Jacobson 2004; Adam 2007; Zamudio Lopez et al. 2012; Evans 2004), we identified 56 design patterns. These identified design patterns can be categorised into 3 categories namely, 1) Architecture 2) Communication and 3) Programming. We list the identified design patterns below. These design patterns are used in Section 4.1.2 to extract the relevant Stack Overflowtags. The list of design patterns extracted from the literature is as follows:

Abstract factory, Active record, Adapter, Bridge, Broker, Builder, Chain of responsibility, Client-server, Command, Composite, Connection Pooling, Data access object (DAO), Decorator, Domain-driven design (DDD), Data transfer object (DTO), Event-driven architecture (EDA), Entity component system (ECS), Facade, Factory method, Flyweight, Front controller, Identity map, Interceptor, Interpreter, Inversion of control (IoC), Iterator, Lazy loading, Mediator, Memento, Microservices, Model view adapter (MVA), Model View Controller (MVC), Multitier, Model View Presenter (MVP), Model View Viewmodel (MVVM), N-Layer, N-Tier, Naked objects, Observer, Peer-to-peer (P2P), Prototype, Presentation Abstraction Control, Proxy, Publish–subscribe, Repository, Representational state transfer (REST), Service locator, Singleton, Service-oriented architecture (SOA), Simple Object Access Protocol (SOAP), Specification, State, Strategy, Template method, Visitor, WCF Message Exchange.

By analysing the literature related to design context (Bi et al. 2018; Carlson et al. 2016; Bedjeti et al. 2017; Kyakulumbye et al. 2019; Belecheanu et al. 2006; Harper and Zheng 2015; Power and Wirfs-Brock 2018; Babar et al. 2009; Tang and Lau 2014; Riaz et al. 2015; Petersen and Wohlin 2009; Groher and Weinreich 2015; Clarke and O'connor 2012) we identified an initial list of keywords and their definitions which we then use in deriving the taxonomy as described in Section 4.3. The list of design context related terms we found is as follows:

Web Application, Distributed Systems, Mobile application, Embedded systems, Finance systems, Transport systems, Databases, Network Context, Hardware Context, Stakeholder Context, Quality Standards, Testing Standards, Availability, Functionality, Maintainability, Performance, Security, Usability.

4.1.2 Data Preparation

We downloaded the publicly available Stack Overflow data dump[1] released in September 2019. The dataset spans 133 months, from 31st July 2008 to 1st of September 2019, containing 45,919,820 posts in total, including questions and answers. For this research, we only use the questions posts' text content as the concerns of DPC is raised in the form of questions text.

The data dump from Stack Overflow has six XML files: `Badges.xml`, `Comments.xml`, `PostHistory.xml`, `PostLinks.xml`, `Posts.xml`, and `Tags.xml`. The `Posts.xml` consists of Stack Overflow posts and as the first step we extract question posts from `Posts.xml` file. This `Posts.xml` file is around 75GB and consists of 8 different post types (i.e. Question posts, Answer posts, Wiki posts etc). Each of these post type can be identified by the `Post Type ID` attribute. By using `Post Type ID` we extract the question posts which has the Post type ID of 1. This results in 18,154,493 question posts.

To reduce the scope of the study to the most relevant posts, we applied a filtering criterion on the question posts. We use the list of design patterns identified in Section 4.1.1 to extract the relevant Stack Overflow tags. A Stack Overflow tag consists of a tag description that explains the intended use of the tag. We analysed the tag descriptions and identified 53 corresponding tags for the identified design patterns, as shown in Table 1.

We group DPs into three groups (architecture, programming and communication) based on their main functions. Architecture DPs are used mainly to help architecture design. Whilst some of them such as *Model View Controller (MVC)* have programming support,

[1] https://archive.org/download/stackexchange

Table 1 Design Pattern Tags used to filter DP questions from Stack Overflow

Architecture	Programming	Communication
activerecord	abstract-factory	connection-pooling
client-server	adapter	front-controller
dao	bridge	mediator
domain-driven-design	builder	publish-subscribe
eda	chain-of-responsibility	wcf
entity-component-system	command-pattern	
identity-map	composite	
microservices	decorator	
model-view-controller	dto	
multi-tier	facade	
mvp	factory-method	
mvvm	flyweight-pattern	
n-layer	interceptor	
n-tier-architecture	interpreter-pattern	
naked-objects	inversion-of-control	
p2p	iterator	
repository-pattern	lazy-loading	
rest	memento	
service-locator	observer-pattern	
soa	prototype-pattern	
soap	proxy-pattern	
specification-pattern	singleton	
	state-pattern	
	strategy-pattern	
	template-method-pattern	
	visitor-pattern	

we deem its purpose architectural. DPs in the communication group also have programmer support but their main purpose is to provide patterns for communication design. The DPs in the programming group serve different purposes, some are about software structure and some about software behavior. At this stage, we do not see any advantages for further classification, so we leave them in the one group as patterns to support software programming.

Certain design patterns that we identified during the literature review phase, such as "action-domain-responder" and "model-view-adapter" did not appear as tags or tag descriptions, hence were not included in the final 'Stack Overflow tags' list. These tags were used to select the question posts that were tagged with a particular design pattern. After applying the tag-based filtering, the Stack Overflow dataset is reduced to 215,942 question posts. Some question posts include code segments in the form of `<pre><code> ... </code></pre>` HTML tags. We remove all code segments and only use the sentences from the question posts to obtain the refined Stack Overflow posts.

4.2 Data Labelling

First, a pilot study is conducted by the first author, and the results are discussed among authors to prepare a labelling guideline. Using the labelling guideline, each annotator initially labels a sample of 50 posts, which is discussed among all three annotators to identify discrepancies and conflicts. Any discrepancies or conflicts are discussed and resolved, the labelling guideline is updated, and the labelling process is started.

The data labelling happens in three stages. In the first stage, the purpose is to consolidate the DPC taxonomy. In this stage, the labels have not been finalised yet, and dynamically grow. We stop the labelling once a stopping criterion is met, which is described in the next sections. Once the taxonomy is consolidated, we have a final, unchanging set of labels which is used in the second stage to label the posts required for training and validating the DPC Miner. The trained DPC Miner is used to select context related posts from the very large Stack Overflow corpus. In the third stage, the context-related posts identified by the DPC Miner are manually labelled according to the detailed taxonomy with context subcategories. This final dataset consisting of 1,500 labelled posts is then used to analyse the relationship between design context and design patterns.

The level of agreement between three annotators is measured using Fleiss' Kappa test (Fleiss 1971), which extends Cohen's kappa (Zaiontz 2021). We calculate Fleiss' Kappa for a sample of 50 posts where all three annotators annotate the sample posts and as design context related or non-design context related. We obtained the kappa value of 0.795, meaning substantial agreement(Landis and Koch 1977) between three annotators.

4.3 DPC Taxonomy

The DPC Taxonomy is derived following an iterative approach as shown in Fig. 1, Step 2. First, an initial keyword-based taxonomy is formulated using the keywords derived from the literature review. This taxonomy is then updated with new terms found during the labelling of Stack Overflow posts. A statistical stopping criterion is used to decide when to stop labelling posts and consider the taxonomy complete. Finally, the taxonomy is consolidated by merging common terms and removing redundant keywords.

4.3.1 Formulation of the Initial Taxonomy and Taxonomy Update

The fist step of generating the initial taxonomy is based on the keywords identified from the DPC literature as explained in Section 4.1.1. We generalised these identified keywords and formulate an initial taxonomy which has two level: the main categories, and the subcategory. Subcategories are formulated using the identified keywords from DPC literature, which are then clustered into the main categories. As an example from subcategories like *Web Application, Mobile Application, Embedded Systems* we formulate a main category called *Application Technology*. Then, we revisit the literature using the main categories and subcategories as search terms to get more insight into possible subcategories and update the draft taxonomy. As an example, we identified *Information Systems* as a possible subcategory for *Application Technology*.

For the second iteration, we employ the identified context categories on Stack Overflow posts and conduct a pilot study by labelling 50 posts to obtain more categories and subcategories that we may have missed and update the taxonomy accordingly. As an example, in this iteration, we found discussions like:

Example 1. I am about to create a Desktop App (with .NET windows forms) ...

where the terms *"Desktop App"* are part of the design context. After discussing among authors, we considered it as a possible subcategory of the taxonomy and introduce a subcategory called *Standalone Systems* under *Application Technology*.

Other examples are:

Example 2. I have upload a small sample application ...
Example 3. We have a large Rails application that is not quite a traditional multi-tenant app ...

which mention the size of the application. When we analysed these posts, there was no category to represent the size of the application, hence it was agreed among authors to add a new category called *Application Size* and two subcategories namely *Small* and *Large*. At the end of this process, we have a non-final draft of the taxonomy which goes through the taxonomy update and consolidation step, described in the next section.

4.3.2 Taxonomy Update and Consolidation

In order to develop a consolidated taxonomy, we use the draft taxonomy created using the keywords (as described in Section 4.3.1) to further label Stack Overflow posts and update it with new context terms that arise while labelling. This process is repeated until a statistical stopping criterion is met.

The objective of the stopping criterion is to decide when to stop analysing the Stack Overflow posts. The common intuition is to stop analysing posts when there is no more unique design context found after analysing n number of posts. However, these unique design context and n number of posts are not known at the time of analysing the posts, and might depend on the distribution of unique design context terms in the output of the analysed data which is also unknown. Therefore, we introduce a statistical estimator UC_P (unique context per batch with size P) to estimate the UC_P distribution and to find n. Moreover, we propose a batch-wise analysis on the estimator as the taxonomy generation is continuous. The benefit of using statistical analysis to estimate the number of Stack Overflow posts needed to analyse before taxonomy consolidation is to implement a systematic approach for taxonomy consolidation and to predict the distribution of the unknown data sample by employing statistical significance.

For each batch of Stack Overflow posts, the mean of the estimator $\overline{UC_P}$ is calculated and the statistical significance of the UC_P is estimated as in (1):

$$ME = t_{(1-\alpha/2)} * ((\frac{1}{\sqrt{|P|}} \sum_{uc \in UC_p} (uc - \overline{uc_p})^2)/\sqrt{|P|}) \qquad (1)$$

where uc_p is the number of unique context terms for post p, uc is the total number of unique context terms identified so far, α is the desired significance level and t is the size of the difference relative to the variation in the batch of the t-distribution. The margin of error ME of the estimator UC_P is checked against a threshold value ME_{th} (e.g. 0.002) for at least n_{TH} (e.g. 10) of occurrences before consolidating the taxonomy.

Algorithm 1 Statistical stopping criterion for taxonomy consolidation.

```
$n_{ME} = 0$;
$n_{UC} = 0$;
$n_{TH} = 10$;
while $n_{ME} \leq n_{TH}$ do
    for $p \in P$ do
        $nc_p = 0$;
        for $new\ unique\ context \in p$ do
            $nc_p$ ++;
            $n_{UC}$ ++;
        end
        $NC \leftarrow$ Append($nc_p$);
    end
    for $nc \in NC$ do
        $uc_p = \frac{n_{UC}}{nc}$ ;
        $UC_p \leftarrow$ Append($uc_p$) ;
    end
    $\overline{uc_p} = \sum_{uc \in UC_p} \frac{uc}{|P|}$ ;
    $ME = t_{(1-\alpha/2)} * ((\frac{1}{\sqrt{|P|}} \sum_{uc \in UC_p} (uc - \overline{uc_p})^2) / \sqrt{|P|})$;
    if $ME \leq ME_{th}$ then
        $n_{ME} + +$ ;
    end
end
```

These steps are listed in Algorithm 1 and can be summarised as follows.

1. Initiate the counter variable (n_{ME}, n_{UC}) to zero (line 1-2).
2. Set a value for the number of iterations required to satisfy the stopping criterion (n_{TH}) to 10 iterations (line 3).
3. For each post p in P count the number of unique context terms (n_{UC}) and context posts (nc_p) (line 5–12). As an example, consider the first batch P_1. For the first post p_1 in P_1, and assume we found the context terms *Web Application*, *Learning Management Systems*, *Databases*, *Performance*. Therefore, for p_1 the value of n_{UC} is 4 as it is the first time all these terms are found, with the nc_p value of 1. Next, for p_2, assume we found a term related to *Distributed Systems* and *Performance*. The term *Performance* is already found in p_1 so we do not consider it as a unique context in p_2. Only *Distributed Systems* is considered as UC and therefore the value is n_{UC} is updated to 5, with the nc_p of 2. Likewise, we calculate the n_{UC} for all 20 posts in P_1.
4. Calculate the value of the estimator (UC_P) for the batch (P) using the equation in line 14.
5. Calculate the mean value of the estimators ($\overline{UC_P}$) (line 17).
6. Obtain the margin of error of the estimator (line 18).
7. If the obtained ME is less or equal to the ME_{th} increment the n_{ME} (line 19-21).
8. Proceed to the next batch until the condition in the line 4 is satisfied.

Next, by following the bottom up approach, the new context terms identified are placed under an appropriate main category to consolidate the taxonomy. As an example, we found context terms like *Operating System* which we placed under the main category of *Platform Context*.

4.4 DPC Miner

The overview of the DPC Miner is shown in Fig. 1, Step 3. DPC Miner uses both unsupervised and supervised learning approaches. The RL model is an unsupervised learning approach. The objective of RL model is to generate a domain specific – DP-related – distributed vector representation from the input text by capturing the semantic and syntactic features. In Supervised learning, we train the model by providing design context labeled posts. For both model training, we use the refined Stack Overflow posts after data preprocessing. Finally, both the representation learning model and supervised learning model are incorporated for further DPC mining. Further, we enhance the DPC miner with collocation generation, which creates additional textual features.

4.4.1 Data Prepossessing

Tokenizing Each Stack Overflow post is broken down into its corresponding words using a well known NLTK[2] package. These word form a single unit of text known as a unigram.

Stopword Removal Natural language text consist of a lot of words that makes the text more readable for humans but are not useful for machine learning, and are considered as noise for automated text analysis, impacting their performance. In NLP research, these words are refereed as stop words (e.g., the, a, an). In the DPC Miner the focus should be on more domain specific keywords which define the content of the text. Therefore, we decided to exclude the stop words from the text.

POS Tagging Natural language is made up of a number of parts of speech such as verbs, nouns, adverbs. In POS tagging, the words in the sentence are assigned with the parts of speech (POS) tags based on the sentence structure. These POS tags are the lexical units of the sentence. We use the Python NLTK package[3] for POS tagging. As a part of DPC Miner features we generate collocations. We use this POS tags in these collocation generation described in Section 4.4.2.

4.4.2 Collocation Generation

Collocations are a combination of unigrams, which are created via tokenizing. The purpose of generating the collocations is to provide the representation model with more insight about the textual content and enhance the feature set. As an example there are words in DPC posts which have more meaning as a set of words (i.e. “model view controller”, “connection pooling”, “web application”) than evaluating them individually. Therefore, we argue that using the collocations can improve the performance of the DPC Miner.

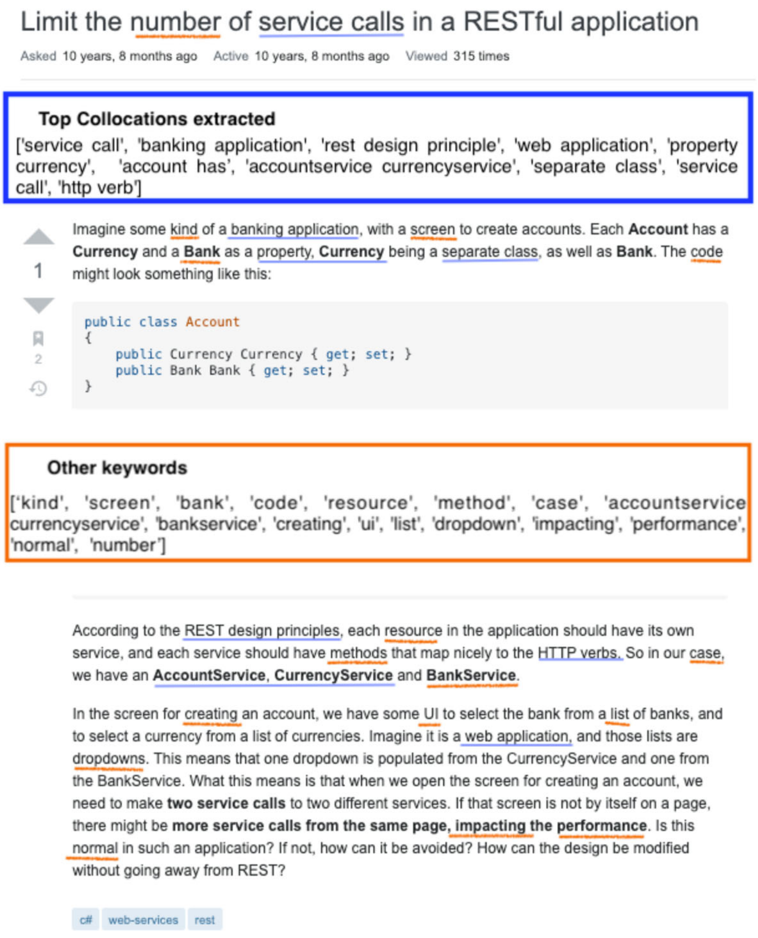
Some lexical units contribute more compared to the others to identifying domain specific terms. Therefore, we perform syntactic filtering in collocation generation. We use the POS tags of the lexical unit in this syntactic filtering. For this study, we focus on nouns,

[2] https://www.nltk.org/api/nltk.tokenize.html

[3] http://www.nltk.org/book/ch05.html

verbs, adjectives, and foreign words in syntactic filtering. We use NN (noun, singular), NNS (noun plural), NNP (proper noun, singular), and NNPS (proper noun, plural) POS tags for the nouns, VB (verb, base form), VBD (verb, past tense), VBG (verb, gerund/present participle), VBN (verb, past participle), VBP (verb, sing. present), VBZ (verb, 3rd person sing. present) POS tags for verbs. For adjective we used JJ (adjective), JJR (adjective, comparative), JJS (adjective, superlative) POS tags and FW (foreign word) POS tag for foreign words.

Next, we collapse the adjacent syntactically filtered lexical units into collocations. Figure 2 shows an example of a Stack Overflow post and the notion of extracted key-terms. The key-terms underlined in blue are the collocations generated by feature generation for the question post. As we can see, the model is capable of generating collocations like "rest design principle", "web application" and "banking application". The key-terms underlined in orange colour are the words identified in the feature generation. DPC Miner identified terms like "performance", "UI", "impacting" and "bank". The output of the collocation



Limit the number of service calls in a RESTful application

Asked 10 years, 8 months ago Active 10 years, 8 months ago Viewed 315 times

Top Collocations extracted

['service call', 'banking application', 'rest design principle', 'web application', 'property currency', 'account has', 'accountservice currencyservice', 'separate class', 'service call', 'http verb']

Imagine some kind of a banking application, with a screen to create accounts. Each **Account** has a **Currency** and a **Bank** as a property, **Currency** being a separate class, as well as **Bank**. The code might look something like this:

```
public class Account
{
    public Currency Currency { get; set; }
    public Bank Bank { get; set; }
}
```

Other keywords

['kind', 'screen', 'bank', 'code', 'resource', 'method', 'case', 'accountservice currencyservice', 'bankservice', 'creating', 'ui', 'list', 'dropdown', 'impacting', 'performance', 'normal', 'number']

According to the REST design principles, each resource in the application should have its own service, and each service should have methods that map nicely to the HTTP verbs. So in our case, we have an **AccountService**, **CurrencyService** and **BankService**.

In the screen for creating an account, we have some UI to select the bank from a list of banks, and to select a currency from a list of currencies. Imagine it is a web application, and those lists are dropdowns. This means that one dropdown is populated from the CurrencyService and one from the BankService. What this means is that when we open the screen for creating an account, we need to make **two service calls** to two different services. If that screen is not by itself on a page, there might be **more service calls from the same page, impacting the performance**. Is this normal in such an application? If not, how can it be avoided? How can the design be modified without going away from REST?

Fig. 2 Example of generated collocations and keywords

generation is key-terms, which are used as the feature in the next phase of representation learning and consist of collocations and words.

4.4.3 Representation Learning Model Training and Feature Extraction

Representation Learning (RL) model is the unsupervised learning component of the DPC Miner. In RL model training, we generate a vocabulary V with size W. V consists of key-terms that are a combination of collocations and uni-grams.

In the RL model, we use V as a feature to generate a dense low-dimensional vector representation based on the assumptions that key-terms (uni-grams and collocations) with similar meaning are present in a similar context space (Harris 1954). We use the continuous skip-gram model proposed by Mikolov et al. (2013a) for the RL model implementation.

The learning objective of the RL model is to learn a vector representation for a given key-term w_i in the V to successfully predict the surrounding key-terms w_{t+j} within a training window of size c. For a given key-term sequence of $w_1, w_2, \ldots, w_T$ the objective is to maximise the average log probability as shown in (2):

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le j\le c, j\ne 0} \log\ p(w_{t+j}|w_t) \tag{2}$$

The value of log p can be defined as in (3), which is the conditional probability defined using the softmax function (Mikolov et al. 2013b):

$$p(w_{t+j}|w_t) = \frac{\exp(v\,'_{w_{t+j}}{}^{\top} v_{w_t})}{\sum_{w=1}^{W}\exp(v\,'_{w}{}^{\top} v_{w_i})} \tag{3}$$

where v_w and $v\,'_w$ are the 'input' and 'output' vector representations of w, and W is the vocabulary size. The output of the RL model is a learned vector v for each key-term w.

For RL Model training, after tuning the hyperparameters, we used vector dimension of 200, window size of 3, and 100 of learning epochs.

As the next step of feature extraction, for a given post, we average the v of key-terms in the post and generate a feature-vector for the post v_{post} to feed into the DPC Miner.

4.4.4 Supervised Learning

Next, we use the supervised learning to train the classifiers. The objective is to learn to classify a given post as a DPC or non DPC post, thus, we use binary classification techniques. We use the DPC labelled dataset to train the classifiers as supervised learning requires a labels in model training.

Further, to decide the most suitable classifier technique for the DPC Miner implementation we implemented eight well-known classifiers using Python Scikit-learn package (Pedregosa et al. 2011). They are Support Vector Classifier (SVC), Random Forest Classifier (RF), Gradient Boosting Classifier (GB), Logistic Regression Classifier (LR), Naive Bayes classifier (NB) , Decision Tree Classifier (DT), K-Neighbours Classifier (KNN), and Multi-layer Perceptron classifier (MLP). Based on the results as shown in the table we selected Support Vector Classification (SVC) to implement the DPC Miner. We use the default parameter setting for each classifier (Pedregosa et al. 2011) for the comparison and only changed the parameter of $class\_weight = balanced$. We change the $class\_weight = balanced$ to ensure the values of that class weights will adjust inverse proportionally to class frequency in the input data hence, to mitigate any bias on results due to the class imbalance. Finally, in the DPC Miner, we use the feature-vectors learned from the RL model as features to feed into the Supervised Learning model to mine more DPC related posts.

4.5 Analysing the Relationship Between Context and Design Patterns

The final step of our methodology is the quantitative analysis of posts mined from Stack Overflow. First, we extend our DPC data corpus with more posts from Stack Overflow which are mined using the proposed DPC miner. Finally, we analyse how each DPC term is discussed with respect to the design patterns.

4.5.1 Mining more Posts

Finally, we use the trained DPC Miner for more context related post mining to expand the dataset. As the input, we provide the DPC Miner with a new set of posts that are not used in DPC Miner training. Next, the trained DPC Miner explores the syntactic and semantic structure of the new set of posts and generates the feature vectors. These feature vectors are used to classify the new set of posts as design context related or not. After the mining process of a new set of posts, we manually evaluate the predicted posts for their design context relatedness, and label them according to the taxonomy.

4.5.2 Quantitative Analysis

To study the relationship between design patterns and context, we perform a cross-case analysis and examine the number and percentage of discussions that are focused on combinations of context and design patterns. For example, we analyse the proportion of discussions that discuss testing standards for each design pattern. This analysis uncovers the main contextual factors for each design pattern.

5 Results

In this section, first we present the DPC Taxonomy which provides an answer to RQ1. We also show the results of the statistical stopping criterion which is used to consolidate the taxonomy. To answer RQ2, we presented the evaluation of the DPC Miner in Section 4.4, which demonstrates that our approach is effective in mining context knowledge related to design patterns. Finally, to answer RQ3, we analyse 1,500 Stack Overflow posts which are manually labelled, and present an overview of how design context is discussed when contemplating which design pattern to use. All the data, replication code, analysis, and results of this paper are available.[4]

5.1 RQ1: DPC Taxonomy

The consolidated DPC Taxonomy is shown in Fig. 3. The first level of the hierarchy structure shows the main categories identified from the design context terms. In the consolidated taxonomy we identified seven main categories: 1) Application Technology, 2) Application

[4] https://github.com/laksW/Mining-and-Relating-Design-Contexts-and-DesignPatterns-from-Stack-Overflow.git

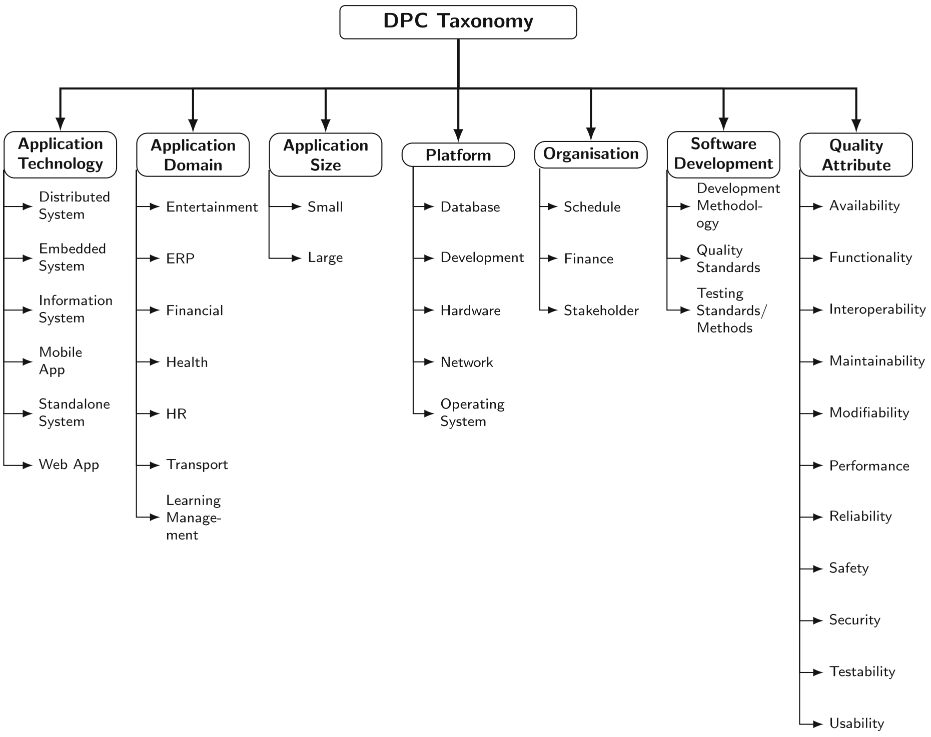


Fig. 3 DPC taxonomy

Domain, 3) Application Size, 4) Platform Context, 5) Organisation, 6) Software Development Context, and 7) Quality Attributes. These first level categories come from various literature as discussed in Section 4.1.1 and from the taxonomy update and consolidation step discussed in Section 4.3.2. Design context terms that we identified are shown as sub categories under each main design context. As an example the design context *Application Technology* has 6 subcategories namely *Distributed System*, *Embedded System*, *Information System*, *Mobile Application*, *Standalone system* and *Web Application*.

To illustrate how these categories are identified we use the following examples which are part of a Stack Overflow question post.

Example 1. I am part of a team building a proprietary Learning Management System for our organization. When a user is enrolled into an online course, ... this is causing some performance issues. I am looking into developing a framework for queuing .... Our platform is deployed via an Azure web app and Azure SQL database. It is written in .NET MVC and we are sending emails via SMTP ...

Example 2. Imagine some kind of a banking application, with a screen to create accounts. Each Account has a Currency and a Bank as a property ...

In Example 1, we can identify four subcategories from the taxonomy, which have been underlined. First, the post mentions a Learning Management System (LMS), which is the domain of the application, hence it falls under *Application Domain* and has a subcategory of Learning Management. Secondly, we can see the discussion raises a concern related to a measurable property of the application by stating *some performance issue*, thus subcategory of *Performance* under *Quality Attribute* is selected. Thirdly, the post mentions technologies related to the application by stating *web app*. Therefore, we identify the subcategory of *Web Application* under *Application Technology*. Finally, we find the term *database*, hence we categorise the post under the *Platform Context*.

In Example 2, the discussion is about *banking application*, which is the domain under which the application is going to be used. However, there is no subcategory called banking application, but we can see a subcategory for *Financial Systems*. Since we already have a suitable subcategory that we can abstract *banking application*, we categorise Example 2 under *Financial Systems*.

For the taxonomy consolidation and the calculation of the stopping criterion, we use a batch size $P = 20$ posts, $\alpha = 0.95$, $ME_{th} = 0.002$, $n_{th} = 10$. Table 2 shows how $\overline{UC_P}$ varies over the number of batches. We stop labelling batches and consider the identified context terms complete when $ME \leq ME_{th}$ and $n_{ME} = n_{th}$. This means that to be confident that we have covered all context terms, we must continue labelling at least 10 new batches once we reach the threshold value for ME. If ME remains at or below the threshold value ME_{th} for n_{th} (in our case, 10) batches of Stack Overflow posts, than we consider the taxonomy complete.

The starting value for $\overline{UC_P}$ (average number of unique context terms per post) is 1.567, and the initial value for ME is 0.518. These two values, however converge quickly and we are able to reach the threshold level of ME after batch 9. For the next nine consecutive

Table 2 Results of the stopping criterion

Batch	n_{UC}	$\overline{UC_P}$	ME	n_{ME}
Batch - 1	15	1.567	0.518	0
Batch - 2	6	0.671	0.059	0
Batch - 3	5	0.459	0.015	0
Batch - 4	1	0.386	0.018	0
Batch - 5	0	0.297	0.011	0
Batch - 6	3	0.254	0.003	0
Batch - 7	2	0.239	0.003	0
Batch - 8	0	0.213	0.005	0
Batch - 9	2	0.196	$\underline{0.002}$	1
Batch - 10	2	0.181	$\underline{0.002}$	2
Batch - 11	1	0.171	$\underline{0.002}$	3
Batch - 12	0	0.161	$\underline{0.002}$	4
Batch - 13	1	0.148	$\underline{0.002}$	5
Batch - 14	0	0.140	$\underline{0.002}$	6
Batch - 15	1	0.133	$\underline{0.001}$	7
Batch - 16	0	0.125	$\underline{0.001}$	8
Batch - 17	0	0.118	$\underline{0.001}$	9
Batch - 18	0	0.111	$\underline{0.001}$	**10**

batches ME stay below the ME_{th} and reach the n_{th}. At this point, using the statistical significance analysis performed, with 95% confidence we can claim that our sample with $\overline{UC_P}$ of 0.111 covers all context terms related to design patterns with an error margin of 0.001.

Finally, we consolidate our taxonomy after exploring 18 batches and the answer to RQ1 is:

Answer to RQ1: From the consolidated DPC Taxonomy we identified seven main contextual factors, namely *Application Technology, Application Domain, Application Size, Platform, Organisation, Software Development, and Quality Attributes*, which practitioners consider when choosing design patterns. For each of these main context categories, we identified subcategories that further classify design context in more detail.

5.2 RQ2: DPC Miner

As explained in Section 4.4 we use eight classification techniques to implement the supervised learning component of the DPC Miner. In addition, we implement an approach to derive collocations, which aims at enhancing the set of features used for supervised learning. To evaluate the DPC Miner and its different components, we also explore the following sub research questions to evaluate if the methods for mining the posts for design contexts and design patterns are effective.

RQ2.1 Which classification technique in DPC Miner performs the best?
RQ2.2 Do collocations improve the performance of DPC classification techniques?
RQ2.3 Is DPC Miner an effective automated technique for mining DPC posts?

For the RQ2.1 and RQ2.2 evaluations we use the data labelled from Data labelling 1 (Step 2.2 in 1) and Data labelling 2 (Step 3.1 in 1). We use relatively similar size data (50% of total for each class) for the both positive (DPC discussions) and negative (non DPC discussions) class.

To answer RQ2.1 we apply 10 fold stratified cross-validation and compare the performances of the 8 classifiers. The objective of using the Stratified cross-validation is to get stratified folds to eliminate class imbalance concerns that can affect to the results in lcassification. These stratified folds are made by preserving the percentage of samples for each class and ensure that each set contains approximately the same percentage of samples of each target class as the complete set. We use the 9 folds (90% of data) to train the technique and the remaining fold is to test the performance of the model. We repeat the process 10 times by rotating the training and test folds.

The performance of the classifiers is measured in terms of traditional classification Accuracy as well as using bipartition-based metrics like Precision, Recall, F1-Score, ROC, AUC. These bipartition-based metrics compute the averaged scores from both classes hence, mitigate the concerns of class imbalance in classification results.

– **Accuracy** is a metric which gives a measure of correct predictions from all predictions, defined as $A(y, \widehat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} I(\widehat{y_i} = y_i)$, where $\widehat{y_i}$ is the predicted value of i^{th} sample, y_i is the corresponding true value, n is the number of samples and I is the indicator function.

- **Precision** is defined as $P = TP/(TP + FP)$, where TP is true positive(i.e. correctly predict the positive class) and FP is false positive (i.e. incorrectly predicts the positive class).
- **Recall** is the ability of predicting all the positive samples, defined as $R = TP/(TP + FN)$, where FN is the number of false negatives (i.e. incorrectly predicts the negative class).
- **F1-Score** is the weighted average of the precision and recall and is defined as $F1 = 2 \times (PXR)/(P + R)$.
- **Receiver Operating Characteristic(ROC) Curve** plots two parameters, True positive rate(TPR) and the false positive rate(FPR). TPR is a synonym for Recall and FPR is calculate as; $FPR = FP/(FP + TN)$.
- **AUC** measures the area under the ROC curve. This represents the relation between true positive and false positive rate.

Table 3 summarises the result for each classification technique implemented in DPC Miner. The best value for each evaluation metric is highlighted in boldface and the second-best value is underlined. According to the results, we can see that all the classifiers perform well and have an accuracy of over 64%. The support vector classification (SVC) outperformed all other classifiers with a considerable performance margin with an accuracy of 0.871, and precision and recall values of 0.815 and 0.880 respectively.

We further analyse the performance of the classifiers using the ROC curve and AUC. ROC curve is the graphical visualisation of the performance of the classification model at all classification threshold (Fawcett and An introduction to 2006). AUC measures the area under the ROC from (0,0) to (1,1) and shows the aggregated measure of performance of all possible classification threshold. AUC ranges from 0 to 1 where the higher the AUC better the performance.

Figure 4 shows the ROC curve and respective AUC values for all the classifier implementations and find that SVC outperforms all other classifiers with an AUC=0.87. Therefore, we conclude that the answer to RQ2.1 is:

Answer to RQ2.1: Support vector classification (SVC) is the best-performing technique in learning design context, and has an accuracy = 0.871, precision = 0.815, recall = 0.880 and AUC = 0.87.

Table 3 Classifier performance

Classifier	Accuracy	Precision	Recall	F1-Score
DT	0.711	0.636	0.663	0.648
GB	0.749	0.688	0.688	0.687
KNN	0.643	0.532	**0.964**	0.685
LR	0.846	0.786	0.847	0.815
MLP	<u>0.852</u>	0.798	0.849	<u>0.822</u>
NB	0.824	0.754	0.837	0.793
RF	0.805	**0.837**	0.641	0.725
SVC	**0.871**	<u>0.815</u>	<u>0.880</u>	**0.846**

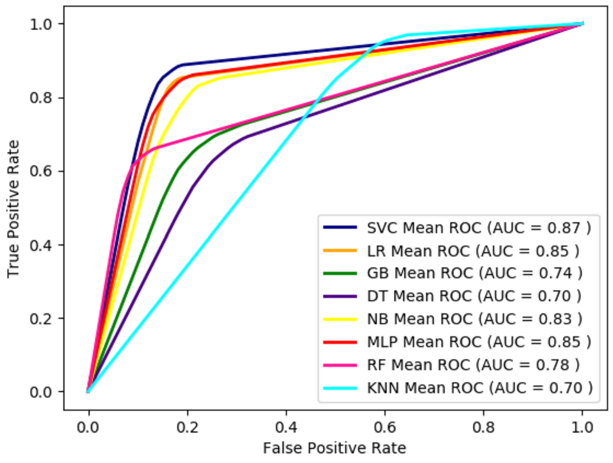


Fig. 4 ROC Curve of eight different classifiers with the AUC score

To answer RQ2.2, we perform an ablation study by evaluating the DPC Miner by removing the collocations. Figure 5 shows the results of the evaluation results in term of Accuracy, Precision, Recall, and F1-Score with and without using collocations.

Across all four evaluation metrics we see that with collocations the evaluation results are higher than without collocations. The accuracy of the DPC Miner without collocations is 0.862 which increased to 0.871 after introducing the collocation. In the same way, Precision increased from 0.803 to 0.815, Recall increased from 0.872 to 0.882, and F1-score increased from 0.835 to 0.846 after using collocations. These results show that collocations identified in feature generation improve the performance of the DPC Miner. Hence, the answer to RQ2.2 is:

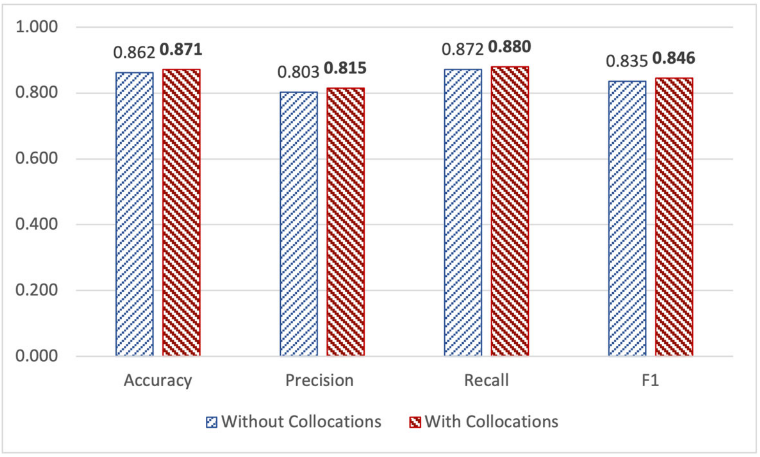


Fig. 5 Performance of the DPC Miner with and without collocations

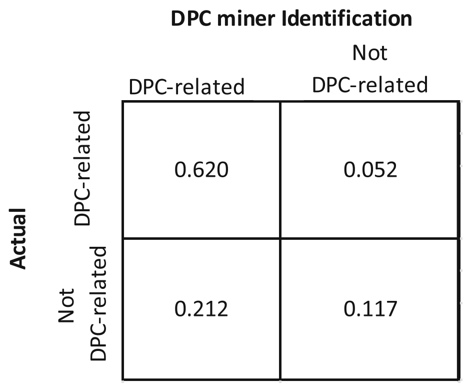
Answer to RQ2.2: Collocations improve the performance of the DPC Miner, increasing precision from 0.803 to 0.815, recall from 0.872 to 0.882, and F1-score from 0.835 to 0.846.

To answer RQ2.3, we use DPC Miner as a mining approach. The objective of the mining approach is to aid the process of collecting more data to expand the DPC related posts. DPC Miner explores the posts and based on the features it automatically suggests the posts as DPC related or not. Then we can only analyse the suggested posts for DPC terms. Therefore, the mining performance is calculated on the correctly classified DPC-related posts since those are the posts we would be using for the processing or further evaluation.

We use the percentage of Precision and Recall to evaluate the DPC Miner performance. From the Precision, we can get an insight on how many relevant posts are mined from the retrieved positive posts since Precision calculates the proportion of the DPC Miner classified positive identifications (true positive and false positive) versus the actual positive (true positive) posts. The mining approach intends to get positive identifications since we use these positive identifications for further analyses. We obtained a value of 74.5% for the Precision. This means that when DPC Miner identifies DPC-related posts 74.5% of the times it was correct when given a new data set with a similar structure. We further calculate the proportion of actual positives and DPC Miner identified correctly (Recall) posts and obtained the value of 92%. This indicates that out of 100 positive posts, DPC Miner is capable of correctly identifying 92 posts. In Fig. 6 we illustrate the normalised confusion matrix of the DPC miner performance.

The experiment set up and the purpose of RQ2.3 is different from the RQ2.1. In RQ2.1 the model is trained and tested on a pre-identified labelled data (DPC related / not DPC related) and evaluated accordingly. In RQ 2.3 the trained DPC Miner is evaluated for its robustness on how it can be utilised as a mining tool to further explore new data. This new data is structurally similar to the data that the model is trained and tested for but it is likely that, this data contains noise and unseen text features that DPC Miner has not previously been trained to recognise.

For experiments in RQ 2.1 and 2.2 the dataset contains 649 DPC-related posts (positive class) from Data labelling 1 (Step 2.2 in Fig. 1) and Data labelling 2 (Step 3.1 in Fig. 1).



Actual	DPC miner Identification: DPC-related	DPC miner Identification: Not DPC-related
DPC-related	0.620	0.052
Not DPC-related	0.212	0.117

Fig. 6 Normalised Confusion Matrix for the DPC Miner performance

With the help of the DPC Miner we further mined Stack Overflow posts to expand the DPC related posts. This mined data is labelled in Data labelling 3 (Step 4.2 in Fig. 1). After the mining, the final dataset consists of 1500 DPC related posts (positive class) in total.

To conclude, the answer to RQ2.3 is:

Answer to RQ2.3: DPC Miner is an effective automated mining technique, and can mine DPC posts with an overall success rate of 74.5% in identifying DPC-related posts.

5.3 RQ3: What Design Context terms are discussed in relation to Design Patterns?

To explore the relations between DP and DC, we empirically analyse the 1,500 posts that were automatically mined using DPC Miner and manually labelled according to the DPC Taxonomy.

5.3.1 Design Context Referenced by Design Patterns

Figure 7 shows a high-level overview of the relationship between DP and DC. The colours in the outer circle are unique to each label (e.g., design pattern (DP) - blue colour, *Platform Context* - orange colour) with the proportion of the circle perimeter representing the frequency of the posts with each label (i.e. DP has the highest posts and *Organisation Contexts* (Org.) has the least number of posts). The line colour in the inner circle represents the colour of the corresponding connected label and the line width represents the strength of the relationship between two labels.

We can see that Quality Attributes (QAs) is the context term with the strongest connection to DPs, followed by *Application Technology*, *Platform Contexts*, and *Software Development Contexts*. QA is a property of a system that can be testable or measurable to indicate how well a system satisfies its stakeholder needs (Bass et al. 2012). In design pattern literature, the effect of patterns on software QAs is one of the most discussed topic (Ampatzoglou et al. 2013). Further, there are several survey studies and systematic literature reviews that investigate the relationship between QAs and design patterns (Galster and Avgeriou 2012; Khomh and Guėhėneuc 2008; Zhang and Budgen 2012; Ampatzoglou et al. 2013). Hence, it is not surprising that we find evidence that QAs are the most important concern when considering design patterns.

Table 4 presents a more detailed analysis of the relationship between DP and DC. In addition to presenting results for each DP individually, we show the summaries for the three main design pattern types 1) Architectural Design Patterns, 2) Communications Design Patterns, and 3) Programming Design Patterns. Each cell shows the number of posts and their proportion which discuss the particular context terms. For example, out of the 216 posts that discuss *Activerecord*, 6 discussions are about *Availability*, constituting 2.8% of the Activerecord-related posts. For each row the highest value is highlighted with boldface.

The most discussed DP type is the Architecture DP with 3,185 mentions, followed by Programming DP type with 286 mentions. The Communication DP type has the least number of mentions with only 34 posts and no posts under *Application Size*, and *Organisation*.

Table 4 High level design contexts referenced by use of design patterns

	Domain	App Size	App Tech	Org	Platform	QA	SW Dev.	Total
Architecture	53|1.7%	26|0.8%	674|21.2%	18|0.6%	721|22.6%	**1053|33.1%**	640|20.1%	3185
Activerecord	6|2.8%	1|0.5%	12|5.6%	1|0.5%	67|31%	**84|38.9%**	45|20.8%	216
Client Server	1|0.5%	4|2.1%	34|18%	–	48|25.4%	**58|30.7%**	44|23.3%	189
DAO	2|5.3%	–	5|13.2%	–	11|28.9%	**13|34.2%**	7|18.4%	38
DDD	8|4.1%	1|0.5%	23|11.7%	2|1%	46|23.4%	**72|36.5%**	45|22.8%	197
EDA	–	–	–	–	1|33.3%	1|33.3%	1|33.3%	3
Identity–Map	–	–	1|33.3%	–	1|33.3%	1|33.3%	–	3
Microservices	4|2%	3|1.5%	28|14.2%	2|1%	49|24.9%	**65|33%**	46|23.4%	197
Multi–Tier	–	1|5%	4|20%	–	5|25%	**7|35%**	3|15%	20
MVC	6|1.2%	3|0.6%	111|21.5%	4|0.8%	112|21.7%	**165|32%**	115|22.3%	516
MVP	1|5.6%	–	4|22.2%	–	4|22.2%	**5|27.8%**	4|22.2%	18
MVVM	2|2.1%	–	15|15.6%	–	20|20.8%	**38|39.6%**	21|21.9%	96
N–Layer	–	–	–	–	–	**1|100%**	–	1
N–Tier	–	–	2|18.2%	1|9.1	1|9.1%	3|27.3%	**4|36.4%**	11
P2P	1|3%	–	6|18.2%	–	12|36.4%	**14|42.4%**	–	33
Repository	–	–	1|14.3%	–	2|28.6%	**3|42.9%**	1|14.3%	7
Rest	18|1.2%	10|0.7%	402|26.6%	5|0.3%	316|20.9%	**484|32.1%**	275|18.2%	1510
Service–Locator	–	– 1|25%	–	1|25%	1|25%	1|25%	4	
SOA	4|3.3%	3|2.4%	25|20.3%	3|2.4%	25|20.3%	**37|30.1%**	26|21.1%	123
SOAP	–	–	–	–	–	1|33.3%	**2|66.7%**	3
Communication	1|2.9%	–	6|17.6%	–	8|23.5%	**13|38.2%**	6|17.6%	34
Conn. Pooling	–	–	1|20%	–	1|20%	**3|60%**	–	5
Mediator	–	–	1|25%	–	1|25%	1|25%	1|25%	4
Publish–Subscribe	1|4.2%	–	4|16.7%	–	5|20.8%	**9|37.5%**	5|20.8%	24

Table 4 (continued)

	Domain	App Size	App Tech	Org	Platform	QA	SW Dev.	Total
WCF	–	–	–	–	**1|100%**	–	–	1
Programming	4|1.5%	5|1.9%	44|16.5%	1|0.4%	65|24.4%	**94|35.3%**	53|19.9%	266
Abstract Factory	1|16.7%	–	–	–	–	**3|50%**	2|33.3%	6
Adapter	–	–	1|20%	–	**2|40%**	1|20%	1|20%	5
Bridge	–	–	1|50%	–	–	1|50%	–	2
Builder Pattern	–	–	–	–	–	1|50%	1|50%	2
Decorator	1|5.9%	–	3|17.6%	–	2|11.8%	**7|41.2%**	4|23.5%	17
DTO	–	–	6|15%	1|2.5%	10|25%	**16|40%**	7|17.5%	40
Facade	–	–	1|10%	–	3|30%	3|30%	3|30%	10
Factory Method	–	–	**6|33.3%**	–	5|27.8%	**6|33.3%**	1|5.6%	18
Interceptor	–	–	–	–	3|33.3%	3|33.3%	3|33.3%	9
Inversion–Control	1|2.6%	1|2.6%	5|12.8%	–	9|23.1%	**13|33.3%**	10|25.6%	39
Iterator	–	–	–	–	1|14.3%	**4|57.1%**	2|28.6%	7
Lazy–Loading	–	–	1|16.7%	–	2|33.3%	**3|50%**	–	6
Memento	–	–	–	–	1|50%	–	1|50%	2
Observer	–	1|14.3%	2|28.6%	–	1|14.3%	2|28.6%	1|14.3%	7
Singleton	1|1.3%	2|2.5%	17|21.5%	–	22|27.8%	**25|31.6%**	12|15.2%	79
Strategy	–	–	–	–	1|14.3%	**4|57.1%**	2|28.6%	7
Visitor	–	–	1|33.3%	–	–	–	**2|66.7%**	3

We can see that in all three DP types, QA is the most discussed context. The trend of having the highest number of DC related discussion among all 3 types of DPs provides empirical evidence that the DC that matters the most when considering DPs is QAs. Only for three patterns – *WCF*, *Memento*, and *Visitor* – we did not find any posts that discuss QAs.

The second most discussed context is *Platform Context*, with 22.6%, 23.5%, and 24.4% of all posts in Architecture, Communication, and Programming-related design patterns respectively. This shows that the *Platform Context* is an equally important concern for all DP types. *Application Technology* is also a frequently discussed context among all DP types, with most discussions focusing on Architecture DP type, and more precisely related to REST with 402 discussions.

Application Domain, *Application Size* and *Organisation* are not as often considered across Communication and Programming DP types, while they are a bigger consideration with Architecture-related DPs with 11 out of 19 Architecture-related DPs containing posts that discuss *Application Domain*. These three contextual factors are largely to do with architecture matters than implementation or communication concerns.

Answer to RQ3: *Quality Attributes* is the context with the strongest connection to DPs, followed by *Application Technology Contexts*, *Platform Contexts*, and *Software Development Contexts*. Certain contextual factors such as *Application Domain*, *Application Size* and *Organisation* are mostly discussed with particular DP types, and are largely related architecture matters.

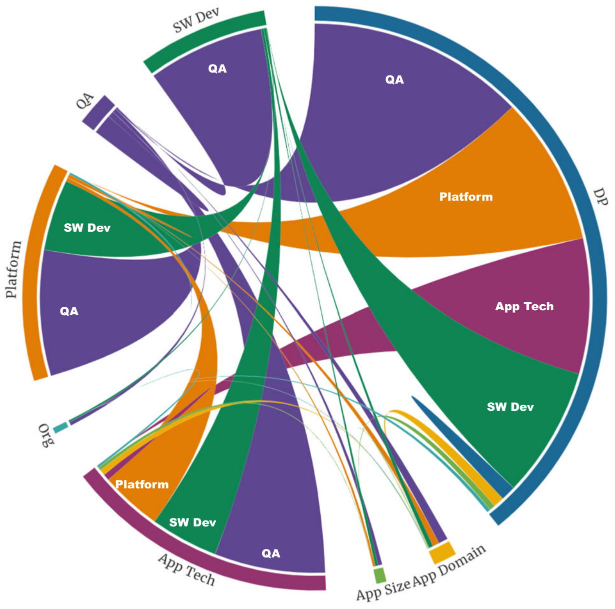


Fig. 7 Design contexts cross referenced with design patterns

5.3.2 Application Technology Context of Design Patterns

The analysis of how *Application Technology* is a concern when considering a DP is presented in Table 5. The majority of the posts are under the Architecture DP type with the *Web Application* context being the most discussed context overall. Examples of discussions from Stack Overflow that mention *Web Application* are:

Example 1. I have a javascript web application almost totally rendered client side. The data is exchanged between client and server using models through a REST interface ...
Example 2. I've been trying to adhere to a strict interpretation of MVC in rebuilding a personal web application at home, but I'm having some issues ...

The next most discussed context is *Mobile Applications*. An example of a post that mentions *Mobile Applications* is:

Example 3. I'm designing a mobile application that communicates with a laptop over LAN where minimal latency is critical ...

The most frequent design pattern under this context is *REST*, which is also the most frequently discussed DP overall (as also shown in the previous section in Table 4). Stack Overflow posts that fall under Communication and Programming design pattern types often mention *Mobile Application*. On the other hand, we do not find any posts that discuss Communication-related design patterns together with *Embedded Systems*, *Information Systems*, and *Standalone Systems*.

5.3.3 Application Domain Context of Design Patterns

Application Domain is not often mentioned in Stack Overflow posts, and the majority of the posts that discuss domain-related factors, focus mainly on *Financial Systems*, as seen in Table 6. One example that discusses *Financial Systems* is:

Example 1. The application is a financial tracking application. I'm stuck at the first page, which is just a list of the bank account transactions for the month ...

It appears that Architecture DP types are related to *Application Domain*, such as *Financial Systems* and *Enterprise Systems*. This is probably because the structure, or architecture, of a system are related to how businesses in a domain operate, like an enterprise is likely to be geographically distributed and thus use *client-server* pattern. *Enterprise systems* is the second most popular context, and we see that posts discussing this context term cover a larger set of design patterns compared to the other *Application Domains*.

The *Application Domain* is not discussed in conjunction with Programming and Communication-related design patterns. This is probably because at the implementation levels, *Application Domains* are irrelevant.

Overall, it is clear that Architecture design patterns are the prominent DP type in the *Application Domain* related discussions, constituting 53 out of 58 mentions.

Table 5 Application technology context referenced by design patterns

	DS	ES	IS	MA	SS	WA	Total
Architecture	21|3.12%	2|0.30%	5|0.74%	133|19.73%	7|1.04%	506|75.07%	674
Activerecord	1|8.3%	–	–	4|33.3%	–	7|58.3%	12
Client Server	2|5.9%	–	–	7|20.6%	2|5.9%	23|67.6%	34
DAO	–	–	–	–	1|20%	4|80%	5
DDD	2|8.7%	–	–	–	–	21|91.3%	23
Identity–Map	–	–	–	–	–	1|100%	1
Microservices	8|28.6%	–	–	2|7.1%	–	18|64.3%	28
Multi–Tier	–	–	–	–	–	4|100%	4
MVC	–	–	–	14|12.6%	2|1.8%	95|85.6%	111
MVP	–	–	–	2|50%	–	2|50%	4
MVVM	–	–	1|6.7%	7|46.7%	–	7|46.7%	15
N–Tier	–	–	–	–	1|50%	1|50%	2
P2P	2|33.3%	–	1|16.7%	–	–	3|50%	6
Repository	–	–	–	–	–	1|100%	1
REST	3|0.7%	2|0.5%	2|0.5%	96|23.9%	1|0.2%	298|74.1%	402
Service–Locator	–	–	–	–	–	1|100%	1
SOA	3|12%	–	1|4%	1|4%	–	20|80%	25

DS=Distributed Systems, ES=Embedded Systems, IS=Information Systems, MA=Mobile Applications, SS=Standalone Systems, WA=Web Applications

Table 5 (continued)

	DS	ES	IS	MA	SS	WA	Total
Communication	**1|16.7%**	–	–	2|33.33%	–	3|50%	6
Conn. Pooling	–	–	–	–	–	**1|100%**	1
Mediator	–	–	–	–	–	1|100%	1
Publish–Subs.	1|25%	–	–	2|50%	–	1|25%	4
Programming	1|2.27%	1|2.27%	1|2.27%	6|13.64%	1|2.27%	34 |77.27%	44
Adapter	–	–	–	–	–	1|100%	1
Bridge	–	–	–	–	–	1|100%	1
Decorator	–	–	–	–	–	3|100%	3
DTO	–	–	–	1|16.7%	–	5|83.3%	6
Facade	–	–	–	1|100%	–	–	1
Factory Method	–	–	–	–	–	6|100%	6
Inversion–Control	–	–	–	–	–	5|100%	5
Lazy–Loading	–	–	–	–	–	1|100%	1
Observer Pattern	–	–	–	–	–	2|100%	2
Singleton	1|5.9%	–	1|5.9%	4|23.5%	1|5.9%	**10|58.8%**	17
Visitor	–	1|100%	–	–	–	–	1

DS=Distributed Systems, ES=Embedded Systems, IS=Information Systems, MA=Mobile Applications, SS=Standalone Systems, WA=Web Applications

Table 6 Application domain context referenced by design patterns

	Ent.	ERP	Fin.	Health	HR	LMS	Trans.	Total
Architecture	15|28.3%	2|3.77%	**28|52.83%**	1|1.89%	3|5.66%	1|1.89%	3|5.66%	53
Activerecord	1|16.67%	–	**4|66.67%**	–	–	–	1|16.67%	6
Client Server	**1|100%**	–	–	–	–	–	–	1
DAO	–	–	**2|100%**	–	–	–	–	2
DDD	–	1|12.50%	**5|62.50%**	1|12.50%	1|12.50%	–	–	8
Microservices	1|25%	–	**3|75%**	–	–	–	–	4
MVC	2|33.3%	–	**3|50%**	–	–	1|16.67%	–	6
MVP	**1|100%**	–	–	–	–	–	–	1
MVVM	1|50%	–	–	–	–	–	1|50%	2
P2P	**1|100%**	–	–	–	–	–	–	1
REST	7|38.9%	1|5.56%	**9|50%**	–	–	–	1|5.56%	18
SOA	–	–	2|50%	–	2|50%	–	–	4
Communication	–	–	–	–	–	–	**1|100%**	1
Publish–Subscribe	–	–	–	–	–	–	**1|100%**	1
Programming	**4|100%**	–	–	–	–	–	–	4
Abstract Factory	**1|100%**	–	–	–	–	–	–	1
Decorator	**1|100%**	–	–	–	–	–	–	1
Inversion–Control	**1|100%**	–	–	–	–	–	–	1
Singleton	**1|100%**	–	–	–	–	–	–	1

5.3.4 Application Size Context of Design Patterns

Application Size is rarely discussed in Stack Overflow posts. In the posts that discuss *Application Size*, we see that the majority of discussions are about small applications. A summary of the results is shown in Table 7.

Most architecture-related design patterns are discussed in the context of *Large* applications, while only one Programming design pattern (Inversion-Control) is covered in *Large* applications. When discussing *Small* applications, we observe a wider coverage of design patterns compared to *Large* applications. We did not find any posts that discuss communication-related design patterns when considering the size of the application. This finding seems to indicate that the size of the application is related to the business and architecture design. Size of application appears to be irrelevant to Programming and Communication design patterns.

Overall, we can see that Architecture DP type discussed more *Application Size* compared to the other two design pattern types. This shows that *Application Size* is more relevant to architecture patterns.

5.3.5 Technology Platform Context of Design Patterns

Most platform related discussions are concerned with *Databases*, and the posts discussing aspects related to *Databases* cover most DPs (more than any other platform-related context). Some examples of such discussion in Stack Overflow posts are:

Example 1. I'm implementing an API using CakePHP3 with a MySQL database ...
Example 2. We are addressing some infrastructure concerns of having so many distinct databases...most urgently, a high number of simultaneous database connections when processing ...

Table 7 Application size context referenced by design patterns

	Large	Small	Total
Architecture	5|19.23%	**21|80.77%**	26
Activerecord	–	1|100%	1
Client Server	1|25%	3|75%	4
DDD	–	1|100%	1
Microservices	1|33.33%	2|66.67%	3
Multi-Tier	–	1|100%	1
MVC	–	3|100%	3
REST	2|20%	**8|80%**	10
SOA	1|33.33%	2|66.67%	3
Programming	2|40%	3|60%	5
Inversion–Control	1|100%	–	1
Observer Pattern	–	1|100%	1
Singleton	–	2|100%	2

As seen in Table 8 apart from *Client Server* and *P2P*, the highest share of posts for each design pattern is the *Database* context. The *Network Context* is mainly considered when discussing the *Client Server* and *P2P* architectural patterns.

Operating system is not very often discussed, with only 8 posts, 6 of which are under the architecture-related design patterns. The *Development* and *Hardware context* are mostly considered under the *REST* design pattern, and there are no posts that discuss development context when considering communication-related design patterns.

Overall, it is clear that *Platform Context* is important to all three categories of design patterns. Although *Database* context is the most popular context in all of the design patterns, the other *Platform Context*s such as *Hardware* and *Network* are also important.

5.3.6 Organisation Context of Design Patterns

Context factors that fall under the *Organisation* are *Schedule*, *Finance*, and *Stakeholder*. Table 9 shows the different organisation-related concerns that are related to the different design patterns. We found only one post that discusses *Finance*, which was related to microservices architectural pattern. On the other hand, *Stakeholder* concerns are the most discussed when it comes to both architecture and programming-related design patterns. One of the example posts is shown in Example 1 where the stakeholder is the customer.

Example 1. If a Customer wants to make an advert in these templates he chooses a template and if its in stock all is good to go ...

The majority of DPs do not appear in Table 9, which indicates that organisation-related aspects are not considered when choosing these design patterns. In particular, we did not find any post under the communication-related design patterns that discusses *Organisation Context*. On the other hand, most of the design patterns that are concerned with *Organisation Contexts* are Architecture design patterns. It is likely that *Organisation Contexts* have an influence on structural design issues rather than implementation issues.

5.3.7 Software Development Context of Design Patterns

There are three contextual factors that are under the *Software development* context category: *Development Methodology*, *Quality Standards*, and *Testing Standards and Methods*.

For the *Development Methodology Context* we considered the factors that are related to different development methodologies (i.e. agile, waterfall). Some of the examples we found discussing *Development Methodology Context* are:

Example 1. ... how can I get a user’s token and send it to client side after authentication on the API ...
Example 2. ... uploading the media for both HMAC authentication middleware ...
Example 3. ... what kind of authentication would you suggest for the REST web services?...

In Example 1, we can see that the post discusses the agile development and state the tension of architecture choice in the Agile environment with dynamic requirements where some members are familiar with old technologies such as SQL procedure, against using

Table 8 Platform context referenced by design patterns

	DB	Dev.	HW	Network	OS	Total
Architecture	**559|77.53%**	19|2.64%	18|2.5%	119|16.5%	6|0.83%	721
Activerecord	**67|100%**	–	–	–	–	67
Client Server	20|41.67%	2|4.17%	2|4.17%	**21|43.75%**	3|6.25%	48
DAO	**10|90.91%**	–	1|9.09%	–	–	11
DDD	**43|93.48%**	1|2.17%	1|2.17%	–	1|2.17%	46
EDA	1|100%	–	–	–	–	1
Identity–Map	**1|100%**	–	–	–	–	1
Microservices	**37|75.51%**	3|6.12%	3|6.12%	5|10.20%	1|2.04%	49
Multi–Tier	**4|80%**	–	1|20%	–	–	5
MVC	**100|89.29%**	3|2.68%	3|2.68%	5|4.46%	1|0.89%	112
MVP	**3|75%**	–	–	1|25%	–	4
MVVM	**18|90%**	–	1|5%	1|5%	–	20
N–TIER	**1|100%**	–	–	–	–	1
P2P	2|16.67%	–	–	**10|83.33%**	–	12
Repository	**2|100%**	–	–	–	–	2
REST	**233|73.73%**	7|2.22%	5|1.58%	71|22.47%	–	316
Service–Locator	**1|100%**	–	–	–	–	1
SOA	**16|64%**	3|12%	1|4%	5|20%	–	25
Communication	**5|62.5%**	–	1|12.50%	2|25%	–	8
Connection Pooling	**1|100%**	–	–	–	–	1
Mediator	**1|100%**	–	–	–	–	1
Publish–Subscribe	2|40%	–	1|20%	2|40%	–	5
WCF	**1|100%**	–	–	–	–	1

Table 8 (continued)

	DB	Dev.	HW	Network	OS	Total
Programming	**52|80%**	1|1.54%	3|4.62%	7|10.77%	2|3.08%	65
Adapter	1|50%	–	–	1|50%	–	2
Decorator	**2|100%**	–	–	–	–	2
DTO	**8|80%**	–	–	2|20%	–	10
Facade	**2|66.67%**	–	–	–	1|33.33%	3
Factory Method	**4|80%**	–	1|20%	–	–	5
Interceptor	**2|66.67%**	–	–	1|33.33%	–	3
Inversion–Control	**8|88.89%**	–	–	1|11.11%	–	9
Iterator	**1|100%**	–	–	–	–	1
Lazy–Loading	**2|100%**	–	–	–	–	2
Memento	**1|100%**	–	–	–	–	1
Observer	**1|100%**	–	–	–	–	1
Singleton	**17|77.27%**	1|4.55%	2|9.09%	1|4.55%	1|4.55%	22
Strategy	**1|100%**	–	–	–	–	1

DB=database, Dev=development, HW=hardware, OS=operating system

Table 9 Organisation context referenced by design patterns

	Schedule	Finance	Stakeholder	Total
Architecture	6|33.33%	1|5.56%	**11|61.11%**	18
Activerecord	**1|100%**	–	–	1
DDD	–	–	**2|100%**	2
Microservices	1|50%	1|50%	–	2
MVC	1|25%	–	**3|75%**	4
N–Tier	**1|100%**	–	–	1
REST	1|20%	–	**4|80%**	5
SOA	1|33.33%	–	**2|66.67%**	3
Programming	–	–	**1|100%**	1
DTO	–	–	**1|100%**	1

newer technologies such as OOP, SOLID, Refactoring. In Example 2, the discussion is about DevOps. DevOps is a software development methodology in that the steps of software development are redefined to streamline programming, testing, and deployment.

For the *Quality Standards Context*, we consider the factors that assure the quality concerns of the system. From the discussions, we found these quality standards contexts are vaguely discussed and concealed within descriptions. Therefore, we have to perceive the whole idea of the post. In most cases, we found that these discussions are around ensuring the quality of the system/application. As an example, in the Example 3, we can see it is trying to question a plausible way to ensure a secure application development, specifically in the areas of user authentication and authorization.

Example 4. One of these applications is CAS which is used for all authorization ...
Example 5. ... is the correct authorization string of the concatenation of the name and token encoded in ...
Example 6. Within my architecture, the resource server and authorization server are the same entity ...

For the category of *Testing Standards/ Methods Context*, we considered the testing methodologies (i.e unit testing) and some of the examples are:

Example 7. Now I have noticed that the performance is dreadful. I started some speed tests with ...

Table 10 presents an overview of the results that show the relationship between design patterns and the *Software Development Context*. We observe that *Quality Standards Contexts* are the most discussed context when considering Architecture and Communication DP types, whereas in Programming DP type, we see a higher number of posts (26 posts, 49.06%) that focus on *Development Methodology Context*. Communication design

Table 10 Software development context referenced by design patterns

	Dev. Methodology	Quality Standards	Testing Stand.	Total
Architecture	217|33.91%	**406|63.44%**	17|2.66%	**640**
Activerecord	12|26.67%	**29|64.44%**	4|8.89%	45
Client Server	9|20.45%	**33|75%**	2|4.55%	44
DAO	3|42.86%	**4|57.14%**	–	7
DDD	11|24.44%	**34|75.56%**	–	45
EDA	–	**1|100%**	–	1
Microservices	21|45.65%	**25|54.35%**	–	46
Multi–Tier	1|33.33%	**2|66.67%**	–	3
MVC	40|34.78%	**72|62.61%**	3|2.61%	**115**
MVP	1|25%	**3|75%**	–	4
MVVM	**11|52.38%**	10|47.62%	–	21
N–TIER	2|50%	2|50%	–	4
Repository Pattern	–	**1|100%**	–	1
REST	98|35.64%	**169|61.45%**	8|2.91%	275
Service–Locator	**1|100%**	–	–	1
SOA	7|26.92%	**19|73.08%**	–	26
SOAP	–	2|100%	–	2
Communication	–	**6|100%**	–	6
Mediator	–	1|100%	–	1
Publish–Subscribe	–	5|100%	–	5
Programming	**26|49.06%**	23|43.40%	4|7.55%	53
Abstract Factory	**2|100%**	–	–	2
Adapter	–	–	**1|100%**	1
Builder Pattern	–	**1|100%**	–	1
Decorator	1|25%	**3|75%**	–	4
DTO	**4|57.14%**	3|42.86%	–	7
Facade	**2|66.67%**	1|33.33%	–	3
Factory Method	**1|100%**	–	–	1
Interceptor	**2|66.67%**	1|33.33%	–	3
Inversion–Control	2|20%	**6|60%**	2|20%	10
Iterator	1|50%	1|50%	–	2
Memento	**1|100%**	–	–	1
Observer	**1|100%**	–	–	1
Singleton	4|33.33%	**7|58.33%**	1|8.33%	12
Strategy–Pattern	**2|100%**	–	–	2
Visitor	**2|100%**	–	–	2

patterns are only discussed in the context of *Quality Standards*, and we do not find any posts that mention these design patterns in the context of *Development Methodology* and *Testing Standards*. *Testing Standards* are the least discussed context, with the majority of the posts focusing on Architecture DP types.

5.3.8 Quality Attribute Context of Design Patterns

Quality attributes (QAs) are an important consideration when applying design patterns, and the relationship between the two has been extensively studied in the literature (Harrison and Avgeriou 2007; Feitosa et al. 2019; Bi et al. 2018). In this section, we analyse how design patterns are discussed with respect to QAs as summarised in Table 11.

Depending on the property of the system that a particular QA addresses, QAs can be divided into two categories: 1) QAs related to the runtime and 2) QAs related to the development time of a system. As an example, *Performance* describes the property of a system at runtime and *Modifiability* describes a property related to the development time of the system. *Security*, on the other hand, is related to both runtime and development time aspects of a system.

By definition, security is a "measure of the system's ability to protect data and information from unauthorised access while still providing access to people and systems that are authorised" (Bass et al. 2012). We observe that security-related discussions cover most phases of software development, including securing physical access to the stored data.

Security is the most discussed QA under the architecture-related design patterns, with 482 posts, constituting 45.77% of the total. In both programming and communication-related design patterns, *Security* is the third most frequently discussed QA. Further analysis of each DP under Architecture DP type reveals that out of 19 patterns only 4 have *Security* as the most discussed QA. Those patterns are *REST*, *Service Locator*, *Multi-tier*, and *MVC*. The majority of security-related posts are about *REST* (302 posts). Indeed, *Security* was one of the QAs that was considered when designing the *REST* constraints (Fielding 2000). In-depth analysis of security-related discussions shows that most frequently used key-terms are *Authentication* and *Authorization*. As an example, we often found discussions related to authentication such as:

Example 1. ... how can I get a user's token and send it to client side after <u>authentication</u> on the API ...
Example 2. ... uploading the media for both HMAC <u>authentication</u> middleware ...
Example 3. ... what kind of <u>authentication</u> would you suggest for the REST web services?...

Further, there are examples on authorisation:

Example 4. One of these applications is CAS which is used for all <u>authorization</u> ...
Example 5. ... is the correct <u>authorization</u> string of the concatenation of the name and token encoded in ...
Example 6. Within my architecture, the resource server and <u>authorization server</u> are the same entity ...

According to Table 11, we can see that *Performance* is the most discussed QA under the programming-related design patterns, with 42 out of 92 posts (45.74%), and communication-related design patterns, with 7 posts out of 13 posts (53.85% of the posts in this category). In architecture-related design patterns, even though *Performance* is the second most discussed QA with 339 (32.19% of the discussions) we can see that the majority

Table 11 Quality attribute contexts referenced by design patterns

	Avail.	Comp.	Funct.	Interop.	Maintain.	Modif.	Perfor.	Relia.	Safe.	Scala.	Secu.	Testa.	Usab.	Total
Architecture	23|2.18%	1|0.09%	17|1.61%	1|0.09%	28|2.66%	89|8.45%	339|32.19%	2|0.19%	42|3.99%	13|1.23%	**482|45.77%**	9|0.85%	7|0.66%	1053
Activerecord	3|3.57%	–	1|1.19%	–	1|1.19%	17|20.24%	**39|46.43%**	–	3|3.57%	–	20|23.81%	–	–	84
Client Server	1|1.72%	–	–	–	1|1.72%	2|3.45%	**30|51.72%**	–	–	–	23|39.66%	–	1|1.72%	58
DAO	–	–	–	–	–	3|23.08%	**7|53.85%**	–	–	–	3|23.08%	–	–	13
DDD	3|4.17%	–	3|4.17%	–	6|8.33%	16|22.22%	**30|41.67%**	–	–	1|1.39%	10|13.89%	1|1.39%	2|2.78%	72
EDA	–	–	–	–	–	**1|100.00%**	–	–	–	–	–	–	–	1
Identity–Map	–	–	–	–	–	–	**1|100.00%**	–	–	–	–	–	–	1
Microservices	7|10.77%	–	4|6.15%	–	2|3.08%	6|9.23%	**22|33.85%**	1|1.54%	2|3.08%	1|1.54%	20|30.77%	–	–	65
Multi–Tier	–	–	–	–	–	1|14.29%	**3|42.86%**	–	–	–	**3|42.86%**	–	–	7
MVC	1|0.61%	–	2|1.21%	–	6|3.64%	22|13.33%	50|30.30%	–	5|3.03%	–	**72|43.64%**	5|3.03%	2|1.21%	165
MVP	–	–	–	–	–	1|20.00%	**3|60.00%**	–	–	–	1|20.00%	–	–	5
MVVM	2|5.26%	–	3|7.89%	–	2|5.26%	3|7.89%	**13|34.21%**	–	2|5.26%	–	11|28.95%	1|2.63%	1|2.63%	38
N–Layer	–	–	–	–	–	**1|100.00%**	–	–	–	–	–	–	–	1
N–Tier	–	–	–	–	1|33.33%	1|33.33%	1|33.33%	–	–	–	–	–	–	3
P2P	1|7.14%	–	–	–	–	–	**7|50.00%**	1|7.14%	–	–	5|35.71%	–	–	14
Repository	–	–	–	–	–	1|33.33%	1|33.33%	–	1|33.33%	–	–	–	–	3
REST	4|0.83%	1|0.21%	4|0.83%	–	7|1.45%	11|2.27%	116|23.97%	–	29|5.99%	8|1.65%	**302|62.40%**	2|0.41%	–	484
Service–Loc.	–	–	–	–	–	–	–	–	–	–	**1|100.00%**	–	–	1
SOA	1|2.70%	–	–	1|2.70%	1|2.70%	3|8.11%	**16|43.24%**	–	–	3|8.11%	11|29.73%	–	1|2.70%	37
SOAP	–	–	–	–	**1|100.00%**	–	–	–	–	–	–	–	–	1

Table 11 (continued)

	Avail.	Comp.	Funct.	Interop.	Maintain.	Modif.	Perfor.	Relia.	Safe.	Scala.	Secu.	Testa.	Usab.	Total
Communication	–	–	–	–	2|15.38%	2|15.38%	**7|53.85%**	–	–	1|7.69%	1|7.69%	–	–	13
Connection Pooling	–	–	–	–	1|33.33%	–	1|33.33%	–	–	–	1|33.33%	–	–	3
Mediator	–	–	–	–	**1|100.00%**	–	–	–	–	–	–	–	–	1
Publish–Subs.	–	–	–	–	–	2|22.22%	**6|66.67%**	–	–	1|11.11%	–	–	–	9
Programming	2|2.17%	–	1|1.08%	4|4.35%	3|3.26%	25|27.17%	**42|45.65%**	–	–	–	13|14.13%	1|1.08%	1|1.08%	92
Abstract Factory	–	–	–	1|33.33%	–	**2|66.67%**	–	–	–	–	–	–	–	3
Adapter	–	–	–	–	–	–	–	–	–	–	**1|100.00%**	–	–	1
Bridge	–	–	–	–	–	–	–	–	–	–	**1|100.00%**	–	–	1
Builder Pattern	–	–	–	–	–	–	**1|100.00%**	–	–	–	–	–	–	1
Decorator	1|14.29%	–	–	–	–	1|14.29%	**3|42.86%**	–	–	–	2|28.57%	–	–	7
DTO	–	–	–	1|6.25%	–	4|25.00%	**8|50.00%**	–	–	–	1|6.25%	1|6.25%	1|6.25%	16
Facade	–	–	–	–	–	–	1|33.33%	–	–	–	**2|66.67%**	–	–	3
Factory Method	–	–	–	–	–	2|33.33%	**4|66.67%**	–	–	–	–	–	–	6
Interceptor	–	–	1|33.33%	–	–	–	1|33.33%	–	–	–	1|33.33%	–	–	3
Inversion–Of–Cont.	1|7.69%	–	–	–	–	5|38.46%	5|38.46%	–	–	–	2|15.38%	–	–	13
Iterator	–	–	–	–	–	2|50.00%	2|50.00%	–	–	–	–	–	–	4
Lazy–Loading	–	–	–	–	–	–	**2|66.67%**	–	–	–	1|33.33%	–	–	3
Observer Pattern	–	–	–	–	–	1|50.00%	1|50.00%	–	–	–	–	–	–	2
Singleton	–	–	–	–	3|12.00%	5|20.00%	**14|56.00%**	–	–	–	3|12.00%	–	–	25
Strategy–Pattern	–	–	–	1|25.00%	–	**3|75.00%**	–	–	–	–	–	–	–	4

of the design patterns in this category (11/19 design patterns) have *Performance* as the main concern. Some posts mention performance explicitly, such as:

Example 7. Now I have noticed that the performance is dreadful. I started some speed tests with ...

Other posts mention performance related metrics, such as throughput:

Example 8. 'If I hit my web application directly - for each server I am able to achieve a throughput of 50 RPS per server..."

In both Programming and Communication DP types *Modifiability* is the second most discussed QA. Even though *Modifiability* does not have a high proportion of all posts, it is discussed in relation to most design patterns. As one person puts it:

Example 9. While I'm attracted by the SOA concept (encapsulating reusable business logic into services) I can't figure out how it's supposed to work if data tables encapsulated ...

This finding indicates that practitioners are often concerned about the runtime properties (i.e., *Performance*) of the system across all three DP types.

Often, the trade-off between QAs is considered. For example, one person writes:

Example 11. I am seeking here for the best way to handle database connection(s), concerning aspects of : maintainability, performance, security and implementation ...

When looking at the different QAs discussed for each design pattern, we observe that *REST* has the highest number of different QAs covered by posts (10 different QAs), followed by *Microservices* (9 QAs), DDD (9 QAs), and SOA (8 QAs).

6 Discussion

Software design impacts on the quality, cost, and timeliness of the final solution (Carlson et al. 2016) and involves a knowledge-intensive problem-solving activity (Babar et al. 2009) with many critical decisions (Papatheocharous et al. 2015). Requirements, contexts, and architecture evaluation are the three main knowledge sources that drive the architecture design decisions (Babar et al. 2009). In this paper, we focus on context. Dybå et al. (2012) present a number of ways that enable the exploration and analysis of contextual impact. Our study falls under three of the categories (Dybå et al. 2012):

- "Study critical events that can punctuate context and make possible research and theory that form part of a larger whole". In our study, we investigate the relationship between context and design patterns, which addresses the gap in our understanding of design context.

- "Collect qualitative data that illuminate context effects and interactions that might affect behavior in a studied setting, or that can aid in making inferences about the situation". We mine contextual factors from qualitative data in Stack Overflow posts that impact the choice of design patterns, which enables us to make inferences on their relationship.
- "Measure multiple dependent variables that can uncover situational context when used in conjunction with one another or explain the gap in meaning". The use of design patterns depends on different contextual factors. In this study, we analyse these dependencies.

Following the guidelines suggested by Dybå et al. (2012), we employed an empirical method to discover the relationships of design patterns by mining expert knowledge from Stack Overflow. This allowed us to understand what contextual factors are important to design patterns. For instance, if a developer wants to use a multi-tier architecture, she would then need to ask what size application, what *Organisation Contexts* is intended because these factors are important to help shape decisions.

A new DPC Taxonomy As mentioned earlier there are several work done in this area and our initial DPC Taxonomy construction is based on the different studies related to design context (Bi et al. 2018; Carlson et al. 2016; Bedjeti et al. 2017; Kyakulumbye et al. 2019; Belecheanu et al. 2006; Harper and Zheng 2015; Power and Wirfs-Brock 2018; Babar et al. 2009; Tang and Lau 2014; Riaz et al. 2015; Petersen and Wohlin 2009; Groher and Weinreich 2015; Clarke and O'connor 2012).

Groher and Weinreich (2015) studied environmental factors that influence architectural decision making. This study was conducted qualitatively by analysing expert interviews regarding the decision making process. The authors interviewed 25 experts from 22 different companies in 10 different countries and identified eight categories of factors that influence the architecture decision making namely: Company Size, Business Factors, Organisational Factors, Technical Factors, Cultural Factors, Individual Factors, Project Factors, and Decision Scope. Further, they analyse these architecture decisions in the aspects of how, when, and by whom the decisions are made in different organisation contexts. From this analysis, the authors show that the context in which an architect works influences the decision-making process. In our work, we use this study to get an insight into the context. However, the focus of this study is different from our work since our focus is on design pattern context and our use case is based on both literature and Stack Overflow discussions with a statistically supported quantitative analysis.

Carlson et al. (2016) proposed a context model to allow the explicit representation of the context in the architectural decision making. The proposed context model is comprised of two levels, emphasising the relevance of the context elements in the process of architecture decision making. The top level of the context model consists of five context elements namely: organisation, product, stakeholder, development method and technology, market, and business. The elements under each type of top-level context are defined based on the literature. Further, the authors distinguished the context based on the usage of the context as internal and external context. In our work, we use the same approach like this work where we formulate the second level of context elements based on the literature in our initial taxonomy generation. However, one difference of our work compared to this work is we explore the context elements beyond the literature by exploring further into Stack Overflow discussions. Another difference is the considered domain of the context elements, where we use the design pattern domain instead of architecture decision making.

In the work done by Bedjeti et al. (2017), authors present a *Context Description Viewpoint (CDV)* to describe the context in the software architecture. The authors performed a systematic literature review and identified four context categories (i.e., platform context, user context, application context, and organizational context). These context categories are used as input to create a list of *Context Constituents*. These *Context Constituents* are defined as pieces of information about the context in which the system operates. The authors state that these *Context Constituents* can aid design decisions making in software architecture. Further, to model these *Context Constituents* a graphical model called *Context Description Viewpoint* was introduced. In our work, we consider these *Context Constituents* and context categories when formulating our initial taxonomy.

The closest work to our study is the preliminary analysis of how developers discuss architecture patterns, quality attributes and design context in Stack Overflow (Bi et al. 2018). The study reports that the most frequently asked design question is of the type "should I use this architecture pattern in this application?", which indicates that context is a notable consideration in Stack Overflow posts. The authors argue that "developers often need to know certain information when designing with architecture patterns, such as the relationships between quality attributes and architecture patterns, characteristics and potential issues of using a pattern". These findings inspired our study, which is focused on discovering new design knowledge from empirical data that is regarded as important by developers when considering design patterns, with particular design contexts.

DPC Miner To classify the posts as DPC related discussion or not DPC Miner uses a classifier. In the classification, one of the concerns that can occur is the class imbalance problem. This happens because the input classes are not equally proportioned hence one class may favor compared to another which might affect the final results of the outcome. In machine learning imbalanced data set is unavoidable. It is the same with the DPC Miner. In the process of selecting the negative class samples (non-DPC related posts) for the DPC Miner training and testing, we tried to minimise this by selecting the same percentage (50% each) class samples for both classes. To select non-DPC-related posts we automatically filtered the Stack Overflow discussion posts that do not contain any keywords or phrases related to DPC. However, since we do not manually inspect all of these non-DP posts we used the following measures to ensure imbalance data does not influence the results of the classifier. First, in the classification setting, we used the $balanced_weight$ to ensure that the class weights are adjusted to the input class frequency. Second, we used Stratified sampling in the evaluation to ensure the class imbalance-related concerns are eliminated by stratified folds. The stratified folds preserve the percentage of samples for each class and ensure that each set contains approximately the same percentage of samples of each target class as the complete set. Finally, apart from the traditional evaluation metrics like the accuracy of the classifier we used bipartition-based (Precision, Recall, F1-Score) metrics that by default compute the averaged scores from both classes. Further, we used measures like AUC (Area Under the ROC Curve) that shows the relation between true-positive rate and false-positive rate.

The use of DPC Miner The DPC Miner is a machine learning model that is trained to mine the design pattern context related posts from Stack Overflow. The purpose of the DPC Miner is to predict the possible DPC related discussions such that the number of posts one has to evaluate can be reduced. As an example, lets assume there is a data set with 10,000 posts. In this case a practitioner can use the DPC miner to predict the DPC related posts. Since we already identified the performance of the DPC miner as 74.5% it is likely to predict around 7,450 posts out of 10,000 correctly classified. In our use case, even though we used the DPC Miner to mine more posts, we manually labelled all the posts since we wanted to evaluate the mining performance. However, in practice, once we use DPC Miner to expand the dataset, we only need to label the predicted values.

Relationships between Design Context (DC) and Design Pattern (DP) We found that of all the DPs, irrespective of type, the most discussed DCs are (a) quality attributes; (b) platforms; (c) application technology and (d) software development (see Fig. 7).

Developers are concerned with how quality attributes of DPs shape a solution, and how quality requirements dictate what DPs can be used. We break down the DPs into 3 categories, architecture, programming and communication, and analyse the relationships between each category and DCs. We find that architecture DPs are most concerned with DCs, having 1053 quality attribute DC related discussions. Whereas quality attribute related discussions only appear in communication and programming DP discussions 13 and 92 times, respectively (see Table 11). Additionally, amongst all the different quality attribute sub-categories, security and performance are by far the most concerned DCs in architecture DPs. Performance is the most concerned DC in programming and communication DPs.

Platform contexts are closely related to architecture design patterns. A vast majority of context discussions, i.e., 721 out of a total of 794, are architecture DP related (see Table 8). It basically means that whenever developers discuss the use of architecture DP, they also discuss platform contexts. Amongst these platform contexts, database and networking are discussed the most. This can mean the two contexts are tricky in design and needed information is unavailable to the developers. Similar to platform contexts, application technology contexts are closely related to architecture DPs. Table 5 shows that 674 out of a total of 724 application technology DC discussions are related to architecture DPs.

The popularity of software development contexts is a surprising finding as one would not normally associate a technical design issue with DP to a software development methodology or process. From the mined data, we found that quality standards is the main context for DPs, especially for architecture DPs (i.e. 640 out of a total of 699 discussions, Table 10). This implies that developers want to know quality implementation of DPs, and part of that is testing. Development methodology is also a concern. Developers want to know how DPs can be used in conjunction with a new development environment that they find themselves in.

The need for mining design pattern knowledge Developers often ask about similar questions such as how to use a DP with a particular technology and what would that mean to certain quality attributes. In example 3 in Section 5.3.2, a developer asked about how using a DP in a mobile app affects communication latency. The repeated appearance of similar questions that have interrelated contexts tells us that there is a need to mine and organise such knowledge to allow developers to easily find development knowledge. This kind of software engineering knowledge gathering and dissemination is a bottom-up data mining approach. It is different from the top-down textbook approach that provides principles and theories. As discussed earlier, design context is about the design environment and this environment is highly dynamic in nature. This environment creates a question on how to extract and organise knowledge for the software development community.

As developers share their development experience in discussion forums such as Stack Overflow, the data mining approach that we propose in this article allows us to mine and relate software engineering knowledge, in this case what design contexts to look out for when using design patterns. The categorisation of DPs and DCs and their relationships provide a structure for developers to follow.

Whilst Stack Overflow at this point in time contains relevant data to allow us to carry out mining for this knowledge, it does not necessarily provide ALL relevant context-DP knowledge. It is likely that some contexts are missing and we do not know about them. A methodology such as the one described in this article can and should be used repeatedly over time and on different forums to mine more knowledge. In the end, it would be difficult to test that the context factors can ever be complete as new contexts can emerge over new situations and time.

Future use of Design Pattern Knowledge Design Pattern knowledge (DPK) can be defined as the relation between DP and DPC. Since DPK is mined and readily available it is now possible that we use this structure to guide developers to find relevant posts within Stack Overflow by indexing the URLs of the posts through DC and DP categorisation. We can build an automated tool to structure link DC-DP relationships to the actual discussion. This can help developers find, with a click of a button, the detailed discussions of developers grouped by any of the relationships that we list in Section 5.3. This facility can greatly help developers find out more about the considerations of using certain DPs.

Further, with the use of a knowledge management tool such as an ontology, we can structure this DPK into both machine and human interpretable format and can allow further access to the identified knowledge.

7 Threats to Validity

Internal Validity We use the Stack Overflow tags to identify posts that are related to design patterns. It is possible that these tags are not accurate. To minimise this threat, we assess the accuracy of the tag during the labelling process, and remove posts that were wrongly labelled with a design pattern tag.

Further, there are few occasions that the identified DC is not semantically referring to the DC. This is because the keywords related to the DC in the posts but it does not really discuss the design context. As an example, the post “I read through Agile Web Development with Rails in order to get up to speed ...” contains the term “Agile Web Development”, which is a software development methodology. However, the term appears because it is the name of the book that he happened to mention in the post. We identify these as false positives and can be a threat to internal validity.

Another threat to the internal validity of the results is the hyperparameter tuning for the machine learning models. We performed a random parameter search for the RL model and for the machine learning algorithms, we used the default parameter settings. It is possible that the performance of the ML algorithms can be improved by further tuning the parameters. However, the DPC Miner achieved good performance even with the default values.

The labelling of the data may involve some bias, as different annotators may have a different understanding of context. To minimise this threat, the data was labelled by three annotators that are knowledgeable with design patterns and familiar with the literature in design context. In addition, we assessed the agreement between annotators using Fleiss’ Kappa test, as discussed in Section 4.2.

External Validity External validity is about how much our results can be generalised. Whilst we have mined Stack Overflow extensively, it does not mean that Stack Overflow contains all the DPs and DCs that are known to the SE community. We did not mine other

Q&A sites and there are DCs and DPs that are never discussed in these public forums. As such, our claims in this research is limited to what we have mined so far.

Construct Validity We select Stack Overflow because of the amount of Q&A posted by developers related to design patterns. It is possible that some of the questions are posted by users instead of developers, however, through a manual inspection, we observe that the vast majority of the posts are by professional programmers. To reduce this threat, we considered only posts that have a positive score. We have high precision of correctly labelling design contexts and design patterns, therefore we have high confidence of the accuracy of the relationships between them. Another construct validity coincide with selection of appropriate evaluation metrics. To mitigate this threat, we used multiple evaluation metrics, such as *Precision*, *Recall*, *F1-Score*, *Accuracy*, and *ROC-AUC* which are well used in many SE related mining and classification approaches (Beyer et al. 2019; Ali et al. 2018; Alreshedy et al. 2018; Mirakhorli and Cleland-Huang 2016; Zanoni et al. 2015). Therefore, we believe the threats to construct validity are minimal.

8 Conclusion

Design contexts that influence the use of design patterns have never been systematically investigated. In this work, we systematically mine design contexts and relate them to design patterns from Stack Overflow. There are three main contributions that this study makes. First, we introduce a consolidated taxonomy of design context related to design patterns, which covers context terms that developers consider when implementing design patterns in practice. Second, we develop an automated approach that successfully mines design pattern context knowledge from Stack Overflow. Third, we present an empirical analysis of the relationships between design context and design patterns, uncovering knowledge about what design contexts are considered with design patterns.

As future work, we can extend our mining method to analyse the relationships between the different context categories to reveal important insights around the interplay of the different contextual factors. For instance, the trade-offs between different quality attributes can reveal the interactions between various design contexts such as quality attributes and the application domain. Furthermore, we can expand this study to analyse user profiles such as the expertise of the user (Yang et al. 2014), the reputation of the user (Sillito et al. 2012)), and technological aspects (Chen et al. 2016) of the posts to obtain insights about the demographics of DP and DPC related posts. One can also analyse the timeline of the mentions of DP and DPC to get an insight into the trend line of DP and DPC over time.

References

Adam (2007) Entity Systems are the future of MMOG development – Part 1. http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/

Ahmad A, Chong F, Shi G, Yousif A (2018) A survey on mining stack overflow: question and answering (Q&A) community. Data Technol Appl 52(2)

Ali I, Asif M, Shahbaz M, Khalid A, Rehman M, Guergachi A (2018) Text categorization approach for secure design pattern selection using software requirement specification. IEEE Access 6:73928–73939. https://doi.org/10.1109/ACCESS.2018.2883077

Allamanis M, Sutton C (2013a) Mining source code repositories at massive scale using language modeling. In: Proceedings of the 10th working conference on mining software repositories, MSR '13. IEEE Press, Piscataway, pp 207–216. http://dl.acm.org/citation.cfm?id=2487085.2487127

Allamanis M, Sutton C (2013b) Mining source code repositories at massive scale using language modeling. In: IEEE international working conference on mining software repositories, (Iim), pp 207–216. https://doi.org/10.1109/MSR.2013.6624029

Alreshedy K, Dharmaretnam D, M German D, Srinivasan V, A Gulliver T (2018) Predicting the programming language of questions and snippets of stackoverflow using natural language processing. arXiv:1809.07954

Ampatzoglou A, Charalampidou S, Stamelos I (2013) Research state of the art on GoF design patterns: A mapping study. J Syst Softw 86(7):1945–1964. https://doi.org/10.1016/j.jss.2013.03.063

Babar MA, Dingsøyr T, Lago P, Van Vliet H (2009) Software architecture knowledge management: Theory and practice. https://doi.org/10.1007/978-3-642-02374-3

Barua A, Thomas SW, Hassan AE (2014) What are developers talking about? an analysis of topics and trends in stack overflow. Empir Softw Eng 19(3):619–654

Bass L, Clements P, Kazmanm R (2012) Software architecture in practice, 3rd edn. Addison-Wesley Professional, Boston

Bedjeti A, Lago P, Lewis GA, De Boer RD, Hilliard R (2017) Viewpoint: Modeling context with an architecture. In: Proceedings - 2017 IEEE international conference on software architecture, ICSA 2017, pp 117–120. https://doi.org/10.1109/ICSA.2017.26

Belecheanu R, Riedel J, Pawar KS (2006) A conceptualisation of design context to explain design tradeoffs in the automotive industry. R D Manag 36(5):517–529. https://doi.org/10.1111/j.1467-9310.2006.00451.x

Bengio Y, Ducharme R, Vincent P, Jauvin C (2003) A neural probabilistic language model. J Mach Learn Res 3(Feb):1137–1155

Beyer S, Macho C, Di Penta M, Pinzger M (2019) What kind of questions do developers ask on Stack Overflow? A comparison of automated approaches to classify posts into question categories. Empir Softw Eng. https://doi.org/10.1007/s10664-019-09758-x

Bi T, Liang P, Tang A (2018) Architecture patterns, quality attributes, and design contexts: How developers design with them. In: Proceedings - Asia-pacific software engineering conference, APSEC, 2018-Decem(December), pp 49–58. https://doi.org/10.1109/APSEC.2018.00019

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3(Jan):993–1022

Borg M, Wnuk K, Regnell B, Runeson P (2017) Supporting change impact analysis using a recommendation an industrial case study in a system: safety-critical context. IEEE Trans Softw Eng 43(7):675–700. https://doi.org/10.1109/TSE.2016.2620458

Buschmann F, Henney K (1993) Pattern-oriented software architecture

Cai X, Zhu J, Shen B, Chen Y (2016) Greta: Graph-based tag assignment for github repositories. In: Computer software and applications conference (COMPSAC), 2016 IEEE 40th Annual, vol 1. IEEE, pp 63–72

Carlson J, Papatheocharous E, Petersen K (2016) A context model for architectural decision support. In: Proceedings - 2016 1st international workshop on decision making in software ARCHitecture, MARCH 2016, pp 9–15. https://doi.org/10.1109/MARCH.2016.6

Casamayor A, Godoy D, Campo M (2012) Functional grouping of natural language requirements for assistance in architectural software design, vol 30, pp 78–86. https://doi.org/10.1016/j.knosys.2011.12.009. http://www.sciencedirect.com/science/article/pii/S0950705111002759

Chattopadhyay S, Nelson N, Nam T, Calvert M, Sarma A (2018) Context in programming: an investigation of how programmers create context. pp 33–36. https://doi.org/10.1145/3195836.3195861

Chen C, Xing Z, Han L (2016) TechLand: Assisting technology landscape inquiries with insights from stack overflow. In: 2016 IEEE international conference on software maintenance and evolution (ICSME). pp 356–366. https://doi.org/10.1109/ICSME.2016.17

Chen T-H, Thomas SW, Nagappan M, Hassan AE (2012) Explaining software defects using topic models. In: Proceedings of the 9th IEEE working conference on mining software repositories, MSR '12. Piscataway, IEEE Press, pp 189–198. http://dl.acm.org/citation.cfm?id=2664446.2664476

Choi J, Choi C, Kim H, Kim P (2011) Efficient malicious code detection using N-gram analysis and SVM. In: Proceedings - 2011 International conference on network-based information systems, NBiS 2011, pp 618–621. https://doi.org/10.1109/NBiS.2011.104

Clarke P, O'connor RV (2012) Towards a comprehensive reference framework, vol 54, pp 433–447. http://doras.dcu.ie/16823/1/ClarkeAndOConnor-Vol54No5-pp433-447.pdf

Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407

Dybå T, Moe NB, Arisholm E (2005) Measuring software methodology usage: Challenges of conceptualization and operationalization. In: 2005 International symposium on empirical software engineering, ISESE 2005, pp 447–457. https://doi.org/10.1109/ISESE.2005.1541852

Dybå T, Sjøberg DI, Cruzes DS (2012) What works for whom, where, when, and why? On the role of context in empirical software engineering. In: International symposium on empirical software engineering and measurement, (7465), pp 19–28. https://doi.org/10.1145/2372251.2372256

Evans E (2004) Domain-driven design: tackling complexity in the heart of software. Addison-Wesley, Boston

Fawcett T, An introduction to ROC (2006) analysis. Pattern Recognit Lett 27(8):861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Feitosa D, Ampatzoglou A, Avgeriou P, Chatzigeorgiou A, Nakagawa E (2019) What can violations of good practices tell about the relationship between GoF patterns and run-time quality attributes?, vol 105, pp 1–16. https://doi.org/10.1016/j.infsof.2018.07.014. http://www.sciencedirect.com/science/article/pii/S0950584918301617

Fielding R (2000) Architectural styles and the design of network -based software architectures. http://search.proquest.com/docview/304591392/

Fleiss JL (1971) Measuring nominal scale agreement among many raters. Psychol Bull 76(5):378–382

Galster M, Avgeriou P (2012) Qualitative analysis of the impact of SOA patterns on quality attributes. In: Proceedings - international conference on quality software, pp 167–170. https://doi.org/10.1109/QSIC.2012.35

Gamma E, Helm R, Johnson R, Vlissides J (1995) Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston

Gokyer G, Cetin S, Sener C, Yondem MT (2008) Non-functional requirements to architectural concerns: ML and NLP at crossroads. In: 2008 the third international conference on software engineering advances. pp 400–406. https://doi.org/10.1109/ICSEA.2008.28

Goodman JT (2001) A bit of progress in language modeling. Comput Speech Lang 15(4):403–434. https://doi.org/10.1006/csla.2001.0174

Groher I, Weinreich R (2015) A study on architectural decision-making in context. In: Proceedings - 12th Working IEEE/IFIP conference on software architecture, WICSA 2015, pp 11–20. https://doi.org/10.1109/WICSA.2015.27

Harper KE, Zheng J (2015) Exploring software architecture context. In: Proceedings - 12th working IEEE/IFIP conference on software architecture, WICSA 2015, pp 123–126. https://doi.org/10.1109/WICSA.2015.22

Harris ZS (1954) Distributional structure. WORD 10(2-3):146–162. https://doi.org/10.1080/00437956.1954.11659520

Harrison NB, Avgeriou P (2007) Leveraging architecture patterns to satisfy quality attributes. In: European conference on software architecture, 4758 LNCS. pp 263–270. https://doi.org/10.1007/978-3-540-75132-8_21

Hindle A, Barr ET, Gabel M, Su Z, Devanbu P (2016) On the naturalness of software. Commun ACM 59(5):122–131. https://doi.org/10.1145/2902362

Hussain S, Keung J, Khan AA (2017) Software design patterns classification and selection using text categorization approach. Appl Soft Comput J 58:225–244. https://doi.org/10.1016/j.asoc.2017.04.043

Jacobson I (2004) Object-oriented software engineering: a use case driven approach. Addison Wesley Longman Publishing Co., Inc., Boston

Kawaguchi S, Garg PK, Matsushita M, Inoue K (2003) Automatic categorization algorithm for evolvable software archive, pp 195–200. https://doi.org/10.1109/IWPSE.2003.1231227

Khomh F, Guėhėneuc YG (2008) Do design patterns impact software quality positively? In: Proceedings of the European conference on software maintenance and reengineering, CSMR, pp 274–278. https://doi.org/10.1109/CSMR.2008.4493325

Kitchenham BA, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, El Emam K, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28(8):721–734. https://doi.org/10.1109/TSE.2002.1027796

Kyakulumbye S, Pather S, Jantjies M (2019) Knowledge creation in a participatory design context: The use of empathetic participatory design. Electron J Knowl Manag 17(1):49–65

Landis JR, Koch GG (1977) The measurement of observer agreement for categorical data. Biometrics 33(1):159–174. http://www.jstor.org/stable/2529310

Linstead E, Rigor P, Bajracharya S, Lopes C, Baldi P (2007a) Mining concepts from code with probabilistic topic models. In: ASE'07 - 2007 ACM/IEEE international conference on automated software engineering, pp 461–464. https://doi.org/10.1145/1321631.1321709

Linstead E, Rigor P, Bajracharya S, Lopes C (2007b) Mining eclipse developer contributions via author-topic models. In: Proceedings - ICSE 2007 workshops: fourth international workshop on mining software repositories, MSR 2007, pp 7–10. https://doi.org/10.1109/MSR.2007.20

Liu D, Jiang H, Li X, Ren Z, Qiao L, Ding Z (2020) DPWord2Vec: better representation of design patterns in semantics. IEEE Trans Softw Eng 5589(c):1–1. https://doi.org/10.1109/tse.2020.3017336

Lukins SK, Kraft NA, Etzkorn LH (2008) Source code retrieval for bug localization using latent Dirichlet allocation. In: Proceedings - working conference on reverse engineering, WCRE, pp 155–164. https://doi.org/10.1109/WCRE.2008.33

Marcus A, Sergeyev A, Rajlieh V, Maletic JI (2004) An information retrieval approach to concept location in source code. In: Proceedings - working conference on reverse engineering, WCRE, pp 214–223. https://doi.org/10.1109/WCRE.2004.10

Marcus A, Rajlich V, Buchta J, Petrenko M, Sergeyev A (2005) Static techniques for concept location in object-oriented code. In: Proceedings - IEEE workshop on program comprehension, pp 33–42. https://doi.org/10.1109/wpc.2005.33

Mikolov T, Chen K, Corrado G, Dean J (2013a) Efficient estimation of word representations in vector space. https://doi.org/10.1162/153244303322533223. arXiv:1301.3781

Mikolov T, Sutskever I, Chen K, Corrado G, Dean J (2013b) Distributed representations ofwords and phrases and their compositionality. In: Advances in neural information processing systems. pp 1–9

Mirakhorli M, Cleland-Huang J (2016) Detecting, tracing, and monitoring architectural tactics in code. IEEE Trans Softw Eng 42(3):205–220. https://doi.org/10.1109/TSE.2015.2479217

Mirakhorli M, Shin Y, Cleland-Huang J, Cinar M (2012) A tactic-centric approach for automating traceability of quality concerns. In: 2012 34th international conference on software engineering (ICSE). pp 639–649. https://doi.org/10.1109/ICSE.2012.6227153

Papatheocharous E, Sentilles S, Petersen K, Shah SMA, Cicchetti A, Gorschek T (2015) Decision support for choosing architectural assets in the development of software-intensive systems: The GRADE taxonomy. In: ACM international conference proceeding series 07-11-Sept. https://doi.org/10.1145/2797433.2797483

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesna E (2011) Scikit-learn: Machine learning in {P}ython. J Mach Learn Res 12:2825–2830

Petersen K, Wohlin C (2009) Context in industrial software engineering research

Poshyvanyk D, Guėhėneuc YG, Marcus A, Antoniol G, Rajlich V (2006) Combining probabilistic ranking and latent semantic indexing for feature identification, pp 137–146. https://doi.org/10.1109/ICPC.2006.17

Power K, Wirfs-Brock R (2018) Understanding architecture decisions in context. In: European conference on software architecture, vol 1. Springer International Publishing, pp 147–155. PowerKenandWirfs-Brock2018UnderstandingContext, https://doi.org/10.1007/978-3-030-00761-4

Riaz M, Breaux T, Williams L (2015) How have we evaluated software pattern application? A systematic mapping study of research design practices. Inf Softw Technol 65:14–38. https://doi.org/10.1016/j.infsof.2015.04.002

Riehle D (2011) Lessons learned from using design patterns in industry projects. In: Lecture notes in computer science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol 6510, pp 1–15. 10.1007/978-3-642-19432-0_1

Rosenfeld R (2000) Two decades of statistical language modeling: where do we go from here? Proc IEEE 88(8):1270–1278. https://doi.org/10.1109/5.880083

Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620. https://doi.org/10.1145/361219.361220

Schmidt DC, Stal M, Rohnert H, Buschmann F (2013) Pattern-oriented, software architecture patterns for concurrent and networked objects, vol 2. Hoboken, Wiley

Sillito J, Maurer F, Nasehi SM, Burns C (2012) What makes a good code example?: A study of programming Q&A in StackOverflow, pp 25–34. https://doi.org/10.1109/ICSM.2012.6405249

Soliman M, Galster M, Salama AR, Riebisch M (2016) Architectural knowledge for technology decisions in developer communities an exploratory study with stackoverflow. In: 2016 13th Working IEEE/IFIP conference on software architecture (WICSA). pp 128–133. https://doi.org/10.1109/WICSA.2016.13

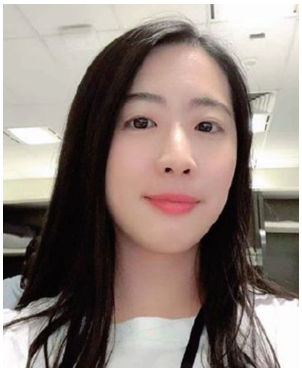
Song F, Croft WB (1999) General language model for information retrieval. In: International conference on information and knowledge management, Proceedings, pp 316–321. https://doi.org/10.1145/319950.320022

Tang A, Lau M (2014) Software architecture review by association. J Syst Softw 88(1):87–101. https://doi.org/10.1016/j.jss.2013.09.044

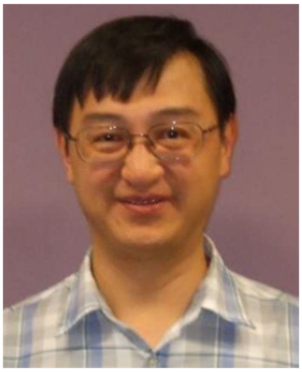
Tang A, Kuo F-C, Lau M (2008) Towards independent software architecture review, pp 306–313. https://doi.org/10.1007/978-3-540-88030-1_25
Thomas SW (2011) Mining software repositories using topic models. In: Proceedings of the 33rd international conference on software engineering, iCSE '11. ACM, New York, pp 1138–1139. https://doi.org/10.1145/1985793.1986020
Tian F, Liang P, Babar MA (2019) How developers discuss architecture smells? An exploratory study on stack overflow. In: Proceedings - 2019 IEEE international conference on software architecture, ICSA 2019, pp 91–100. https://doi.org/10.1109/ICSA.2019.00018
Tian K, Revelle M, Poshyvanyk D (2009) Using latent dirichlet allocation for automatic categorization of software. In: Proceedings of the 2009 6th IEEE international working conference on mining software repositories, MSR 2009, pp 163–166. https://doi.org/10.1109/MSR.2009.5069496
Velasco-Elizondo P, Marin-Piña R, Vazquez-Reyes S, Mora-Soto A, Mejia J (2016) Knowledge representation and information extraction for analysing architectural patterns. Sci Comput Program 121:176–189. https://doi.org/10.1016/j.scico.2015.12.007
Washizaki H, Ogata S, Hazeyama A, Okubo T, Fernandez EB, Yoshioka N (2020) Landscape of architecture and design patterns for IoT systems. IEEE Internet Things J 7(10):10091–10101. https://doi.org/10.1109/JIOT.2020.3003528
Xu B, Ye D, Xing Z, Xia X, Chen G, Li S (2016) Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In: Proceedings of the 31st IEEE/ACM international conference on automated software engineering - ASE 2016, (Id 510357). pp 51–62. https://doi.org/10.1145/2970276.2970357. http://dl.acm.org/citation.cfm?doid=2970276.2970357
Xu B, Xing Z, Xia X, Lo D (2017) AnswerBot: Automated generation of answer summary to developers' technical questions. In: ASE 2017 - Proceedings of the 32nd IEEE/ACM international conference on automated software engineering, pp 706–716. https://doi.org/10.1109/ASE.2017.8115681
Yang J, Tao K, Bozzon A, Houben G-J (2014) Sparrows and owls: Characterisation of expert behaviour in stackoverflow. In: International conference on user modeling, adaptation, and personalization. Springer, pp 266–277
Zaiontz C (2021) Real statistics using excel. real-statistics.com/reliability/interrater-reliability/fleiss-kappa/
Zamudio Lopez SA, Santaolaya Salgado R, Fragoso Diaz OG (2012) Restructuring object-oriented frameworks to model-view-adapter architecture. IEEE Latin Am Trans 10(4):2010–2016. https://doi.org/10.1109/TLA.2012.6272488
Zanoni M, Arcelli Fontana F, Stella F (2015) On applying machine learning techniques for design pattern detection. J Syst Softw 103:102–117. https://doi.org/10.1016/j.jss.2015.01.037
Zhang C, Budgen D (2012) What do we know about the effectiveness of software design patterns? IEEE Trans Softw Eng 38(5):1213–1231. https://doi.org/10.1109/TSE.2011.79
Zhang WE, Sheng QZ, Lau JH, Abebe E (2017) Detecting duplicate posts in programming QA communities via latent semantics and association rules. pp 1221–1229. https://doi.org/10.1145/3038912.3052701
Zhang Y, Witte R, Rilling J, Haarslev V (2006) Ontology-based program comprehension tool supporting website architectural evolution. In: 2006 Eighth IEEE international symposium on web site evolution (WSE'06). pp 41–49. https://doi.org/10.1109/WSE.2006.15

Laksri Wijerathna is a Ph.D. Candidate in the Faculty of Information Technology, Monash University, Australia. In 2014, she received an MSc in Artificial Intelligence from the University of Moratuwa, Sri Lanka, and in 2011, she graduated from the Sri Lanka Institute of Information Technology with a first-class Honours in Information Technology. Her research interests include mining software repositories, software design pattern knowledge management, natural language processing, machine learning, and empirical studies in software engineering.

Aldeida Aleti is an Associate Professor at the Faculty of Information Technology, Monash University in Australia, and the Associate Dean of Engagement and Impact. Aldeida's research is in the area of search-based software engineering (SBSE), with a particular focus on what makes software engineering problems (design, testing, program repair) hard to optimise and designing approaches that make it easier to apply SBSE techniques to new problems. Aldeida has published more than 50 papers in top optimisation and software engineering venues, and has served as PC member and organising committee at both SE and optimisation conferences, such as ASE, ICSE, GECCO, FSE, SSBSE. Aldeida has attracted more than 2.5M in competitive funding to conduct research in the areas of fairness testing of ML-based healthcare systems and for developing search based methods for testing autonomous vehicles. Aldeida was awarded the prestigious Discovery Early Career Researcher (DECRA) Award from the Australian Research Council, and she has received multiple "best paper" and "best reviewer" awards.



Tingting Bi candidate in the Faculty of Information and Technology, Monash University, Australia and the School of Computer Science, Wuhan University, China. She was a visiting Ph.D. candidate at the Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia. Her current research interests include software architecture, empirical software engineering, natural language processing, and machine learning. She has published several articles in peer-reviewed international journals and conferences.



Antony Tang is Adjunct Professor in Swinburne University of Technology, Australia and VU Amsterdam, The Netherlands. He received a PhD degree in Information Technology from Swinburne in 2007. Prior to being a researcher, he had spent many years designing and developing software systems. His main research interests are software architecture design reasoning, software development processes, and software architecture knowledge management.