Architecture Patterns, Quality Attributes, and Design Contexts: How Developers Design with Them?

Tingting Bi [a,b], Peng Liang [a*], Antony Tang [b]
[a] School of Computer Science, Wuhan University, Wuhan, China
[b] Faculty of Science, Engineering and Technology, Swinburne University of Technology, Melbourne, Australia
bi_tingting@whu.edu.cn, liangp@whu.edu.cn, atang@swin.edu.au

***Abstract*—The use of architecture and design patterns have impacts on the quality attributes of a system, and the application of patterns depend on design contexts. There are complex interdependent relationships between them. In this study, we explore how developers use architecture and design patterns with respect to quality attributes concerns and design contexts. We extracted pattern related posts from Stack Overflow and analyzed the architecture discussions. Our study reveals what contexts and quality attributes developers consider when using architecture patterns, and we have identified new and previously unknown relationships between these design elements. These findings can enhance developers' knowledge when they design with architecture patterns, quality attributes, and design contexts.**

Keywords—Architecture Pattern; Design Context; Quality Attribute; Empirical Study

I. INTRODUCTION

Architecture patterns contain a set of rules of how to structure a system into components and connectors [6]. They also represent common ways in which decisions deal with certain aspects of an architecture design [21]. One of the major objectives of using architecture patterns is to structure systems with predictable Non-Functional Requirements (NFRs) [4]. NFRs are important in large-scale software systems and they can be specified as quality attributes (QAs) [3]. Architects and designers need to ensure that both functional and non-requirements are met by the design [1]. There are various architecture and design patterns available for use [7]. When selecting an architecture pattern, quality attributes and their trade-offs need to be considered but in practice they are often omitted.

Design contexts influence a system design in many ways such as developmental, technological, business, operational, social, and other influences [13]. A system of similar functionalities can work differently in different contexts [14][19]. Whilst design contexts are important to making design decisions, the contexts of a system are often ignored and some of the design context may not be explicitly captured in the requirements documents [2]. There is limited research on design contexts (e.g., [2][14]), and there is no standard definition of this concept in the ISO 42010 standard [26].

Software development questions and answers (Q&A) sites (e.g., Stack Overflow, R community, and GitHub) gather knowledge that covers a wide range of topics. These sites allow developers to share experience, offer help, and learn new techniques [5]. Stack Overflow (SO) is one of the most famous and popular online Q&A forums. It contains millions of posts by tens of thousands of developers [8]. SO provides functions such as resurrecting and editing posts that can be inactive for long periods. It supports up voting competing answers and users can earn reputation points by posting interesting questions and answers [10]. Recent studies show that developers and architects use social media to discuss architecture-relevant information (e.g., features and domain concepts) [17][25]. In this work, we took advantage of the architecture and design discussions available in SO. We conducted an investigation of pattern posts in SO to gain insights on how developers consider architecture patterns with respect to quality attributes and design contexts. We are motivated to investigate this topic because there are scarcely any works that provide practical insights to help developers use design and architecture patterns with QA considerations in particular contexts. Often problem domains dictate whether a pattern can be used practically. For instance, financial systems can take advantage of modifiability in MVC pattern but such a system also needs to care about performance which MVC would compromise.

There are two main contributions of this work. First, developers who look for information on how to apply architecture patterns in terms of the QAs are also concerned about specific design contexts. For example, the most frequently asked type of design questions is "*should I use this architecture pattern in this application?*". Second, developers often need to know certain information when designing with architecture patterns, such as the relationships between QAs and architecture patterns, characteristics and potential issues of using a pattern. We identified and listed these considerations (in TABLE VIII) through analyzing the SO posts. It helps developers to recognize some of the QAs and the contexts that need to be considered when using architecture and design patterns.

* Corresponding author
This work is partially funded by the NSFC under Grant No. 61472286 and the China Scholarship Council.

978-1-7281-1970-0/18/$31.00 ©2018 IEEE
DOI 10.1109/APSEC.2018.00019

This paper is organized as follows: Section II discusses related work. Section III describes the methodology. Section IV presents the results of our study. Section V and Section VI discuss the findings and threats to the validity of this study respectively. Finally, Section VII concludes the work with future directions.

II. Related Work

This section introduces the existing work of three design elements (i.e., architecture pattern, quality attribute, and design context) and the methodology (i.e., conceptual model and analysis of developers' posts) employed in our study.

A. Architecture Patterns

Me *et al*. [11] conducted a systematic literature review on the interaction between architectural patterns and quality attributes. They identified four areas by analyzing the selected studies: architecture decision making, architecture patterns, quality attributes, and the interaction between architecture patterns and quality attributes. Harrison and Avgeriou [12] organized a body of knowledge in a way that is accessible and informative for architects, in order to support an architectural decision making process. The authors explored the limitations on the use of architecture patterns in large-scale industrial application, and analyzed several architecture patterns with respect to their impact to key quality attributes. Velasco-Elizondo *et al*. [7] presented an Information Extraction (IE) technique with an ontology for analyzing architectural pattern descriptions with respect to specific quality attributes. An ontology that contains two sub-ontologies was predefined. One is an English grammar-based ontology, which is further categorized into promotes verb, modal verb etc. The other is a performance ontology that defines performance-specific concepts (e.g., security and throughput). Entity extraction and the ontology were used to identify the relationships between architectural patterns and quality attributes in architectural pattern descriptions. This work aims to help inexperienced architects with determining whether specific quality attributes are promoted or inhibited, which is useful for patterns selection. Ali Babar [28] extracted architectural information from architecture pattern descriptions. He proposed a framework with two templates that can effectively capture the relationships between scenarios and quality attributes. This work can help to improve the process of architecture design decision making and evaluation by providing architectural information in an informed format (i.e., templates).

These works are similar to our work that focuses on the relationships between architecture patterns and QAs, while we further analyzed the relationships from the perspectives of design concerns, design problems, and design solutions. In addition, we considered design contexts as a vital factor that influences the interaction between architecture patterns and quality attributes.

B. Quality Attributes

Kim *et al*. [16] presented a systematic quality-driven approach for embodying NFRs into architecture through using architectural tactics. The authors used feature modeling to represent architectural tactics, and applied Role-Based Meta-modeling Language (RBML) to define natural correspondence between the architectural tactics. Their approach can help developers on selecting architectural tactics based on a given set of NFRs, and further analyze the relationships between architectural tactics in terms of availability, performance, and security. In addition, the authors demonstrated a tool with an example for instantiating a composed tactic to generate an initial architecture of the stock trading system. Ameller *et al*. [15] conducted an online survey to explore the role of quality attributes when designing service-based systems. The results of their survey indicate that dependability is the most important QA in a service-based system. In addition, the authors summarized three types of important methods (i.e., architectural decision, architecture pattern, and technology) that can be used to address quality attributes, and found that most of QAs tend to be addressed by ad hoc decisions, rather than architecture or design patterns and technologies.

These works explored the relationships between QAs and various architectural decisions, while in this work we focus on the interactions between QAs and architecture patterns.

C. Design Contexts

Bedjeti *et al*. [2] presented design context viewpoint to capture and describe the contextual elements in architecture. In their work, the authors identified four context categories of the viewpoint (i.e., platform context, user context, application context, and organizational context), then they evaluated and refined the viewpoint by two case studies. Their work can help architects to identify, capture, and analyze contextual knowledge in architecture design. Harper and Zheng [13] proposed a systematic process for identifying and documenting design context to support architectural decisions making. They also proposed a framework in terms of who, what, when, where, why, and how to explore the design contexts of a system and its environment. The results of their work show that the proposed process can drive architectural decisions. Petersen and Wohlin [19] provided a checklist for documenting design contexts in six aspects (i.e., product, processes, practices, techniques, people, organization, and market), and each context facet comprises a set of context elements. The authors then conducted a literature review of industrial studies to investigate to what degree industrial case studies covered the context facets identified in the checklist. Their work can help researchers and practitioners in making informed decisions concerning which parts of the context to be included and not to include. To the best of our knowledge, there has been no investigation on how design

contexts interact with architecture patterns and QAs in terms of design, which is the focus of this work.

D. Conceptual Model

Tang and Lau [14] presented an architecture rationale conceptual model, called Design Association Theory or DAT, based on the ISO/IEC 42010 to illustrate three main design elements (i.e., design concerns, design problems, and design solution options) and verified architecture designs by checking the associations of these design elements. DAT underpins this study and describes architecture as a set of architecture design decisions. In [22], Jansen and Bosch defined a conceptual model for architecture design decisions. Their model comprises several elements, such as design decision, solution, and architectural modification. The heart of these models is the problem element, which describes the architectural design problems together with the motivation and cause elements. The solution element describes the proposed solutions to solve a problem.

E. Analysis of Developers' Posts

Soliman *et al*. [17] analyzed the posts of Stack Overflow to identify architecture-relevant and technology-related knowledge. The authors queried the posts using the terms related to middleware technologies, manually classified the posts into architecture-relevant posts (ARPs) and programming posts, and further classified ARPs into three sub-types (i.e., solution synthesis, solution evaluation, and multi-purpose). The results show that the identified ARPs types could be used to support typical architecting activities. For example, the ARPs that belong to solution evaluation embody important factors for taking architecture design decisions. Barua *et al*. [18] used Latent Dirichlet Allocation (LDA) to analyze the main topics in developer discussions, as well as the relationships between them and the trends of the topics. Forty topics were discovered from Stack Overflow that range widely from jobs to version control to C# syntax. In addition, the authors found that the questions in some topics lead to discussions in other topics, and the trends of several topics (e.g., web development, mobile application, and MySQL) are increasing over time. This work can help developers to understand what topics are discussed and which topics are popular. These works motivate us to extract the knowledge related to QAs, architecture patterns, and design contexts, and further explore the relationships between them using Stack Overflow.

III. METHODOLOGY

We conducted an exploratory study [29] and used a conceptual model (see Section III.A) to define the research questions (in Section III.B) for identifying and extracting the empirical evidence of how developers consider QAs, architecture patterns, and design contexts. The detailed process of data collection and analysis is also elaborated (see Section III.C) in this section.

A. Employed Conceptual Model

We adopted DAT as the conceptual model [14] for decomposing and framing our research questions, which cover design concern, design problem, and design solution aspects of employing architecture patterns to address QAs in certain design contexts (see Fig. 1).

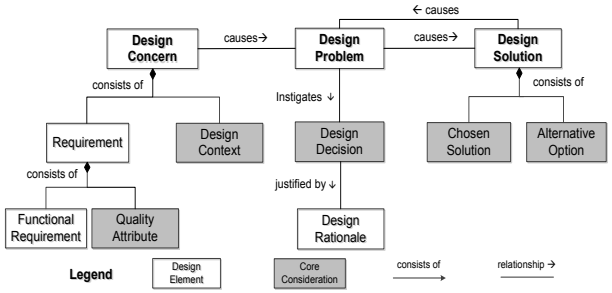

Fig. 1. A conceptual model for design elements.

Design Concern: Design concerns comprise system considerations (i.e., requirements and design contexts). Requirements comprise Functional Requirements (FRs) and **Quality Attributes** (QAs). **Design contexts** comprise the knowledge about the environment that a system is expected to operate and execute [2].

Design Problem: Design problems arise when developers address design concerns and identify the problem space. We explore what common design problems the developers face when they employ **architecture patterns** with respect to **QAs** and **design contexts**. The problems are extracted from the questions that developers ask. The problems can be extracted from the questions that developers ask.

Design Solution: Given a design problem, design solutions and alternative options are discussed by developers. We identify from the SO posts the **design solutions** that arise from the use of **architecture patterns** to solve problems with respect to **QAs** and **design contexts**.

B. Research Questions

The goal of this study is to analyze the use of architecture patterns, and the consideration of quality attributes and design contexts in software design. The perspectives of the analysis are design concerns (e.g., what sort of systems make use of a pattern?), design problems (e.g., what kind of design problems do developers face when they use a pattern?), and design solutions (e.g., how do developers solve the problem?). Such characterization may shed lights on the relationships between architecture patterns, quality attributes, and design contexts. We formulated the RQs in TABLE I from the perspectives provided in the conceptual model (Fig. 1).

C. Data Collection and Analysis

We analyzed Stack Overflow because it contains realistic design dialogues between developers. An overview of the data collection (**Step 1**) from Stack Overflow and analysis (**Step 2**) in this study is shown in Fig. 2.

TABLE I. RESEARCH QUESTIONS AND THEIR RETIONALE

	Research Question	Rationale
Design Concern Aspect	**RQ1**: With respect to using architecture patterns, what QAs and design contexts are developers concerned with?	This RQ provides an overview of the role of QAs and design contexts when developers employ architecture patterns.
	RQ1.1: What QAs do developers consider when they use architecture patterns in a design?	Different QAs often have different significance in different design situations. A quality attribute is a measurable or a testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders. There is a set of QAs, and developers may have more concern about certain QAs than others when they employ architecture patterns. This RQ explores popular QAs considered by developers when they employ certain architecture patterns.
	RQ1.2: What design contexts do developers consider when they employ architecture patterns with respect to QAs?	Design context is the setting and circumstances of all influences upon a system, including developmental, technological, business, operational, social, and other influences [26]. Design contexts are indispensable ingredients that can drive the architecture design of a system. This RQ aims to identify what design contexts are considered when developers use certain architecture patterns with respect to QAs.
Design Problem Aspect	**RQ2**: What are the common design problems developers asked when they employ architecture patterns with considerations of both QAs and design contexts?	Problem space exploration is important, the more structured the problem space is, the more rationally the approach can be taken by researchers and practitioners [20]. Design problems are raised to address design concerns, which is one of the important design activities. This RQ helps to explore what the common design problems developers seek to resolve when they consider architecture patterns, QAs, and design contexts.
Design Solution Aspect	**RQ3**: What are the typical considerations of employing architecture patterns to satisfy QAs in certain design contexts?	A design must have logical causes between a solution and the problems (concerns). Developers may provide different design solutions based on their experience in terms of the same design problems. With this RQ, we identify the typical considerations developers have when they face the design problems of applying architecture patterns with respect to QAs and design contexts.

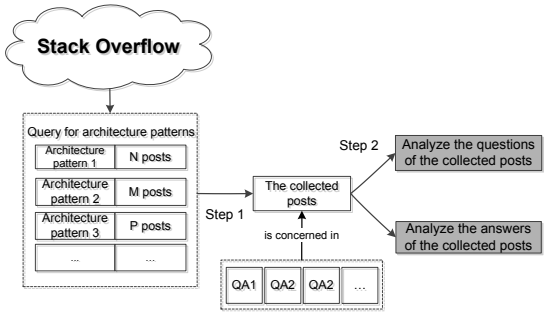

Fig. 2. An overview of data collection and analysis.

Step 1: Searching and collecting candidate posts

Step 1.a. We used the keywords related to popular architecture patterns that are defined in relevant literatures [1][27] to search the architecture pattern posts. We queried the titles, questions, answers, and tags of the posts. The search terms are listed as follows:

Model-View-Controller, *MVC*, *Front Controller*, *Layered*, *Blackboard*, *Pipe & Filer*, *Repository*, *Broker*, *Presentation Abstraction Control*, *PAC*, *Service-Oriented Architecture*, *SOA*, *Client Server*, *Action-Domain-Responder*, *Naked Objects*, *Peer-to-peer Pattern*, *Publish Subscribe Pattern*, *Shared Data pattern*, *Entity-Component System Pattern*, *Mapreduce Pattern*, *Multi-tier Patter*, *Batch Sequential*, *Micro-service.*

Step 1.b. We manually checked candidate architecture pattern posts that were retrieved in **Step 1.a**. We included the architecture pattern posts if they concern one of the QAs. About the taxonomy of QAs, we adopted the ISO 25010 standard that defines eight high-level QAs [9], i.e., Maintainability, Functional Suitability, Security, Performance, Portability, Usability, Reliability, and Compatibility.

Step 2: Analyzing the collected posts

We employed descriptive statistics and constant comparison methods to analyze the collected posts. Specifically, we used descriptive statistics to analyze quantitative data (i.e., the frequency of quality attributes, architecture patterns, and design contexts) and used the constant comparison to analyze qualitative data (i.e., design problems and solutions). Constant comparison is a continuous process for verifying the generated categories [20]. Two steps were conducted in the data analysis.

Step 2.a. We analyzed the questions of the collected posts. Through analyzing the questions, we explore what design concerns (i.e., QAs and design contexts) and common design problems developers may have. We extracted QAs based on the classification of QAs described in Step 1.b. We extracted design contexts based on the categories from [2] and [19], described in Section II.C. We analyzed the discussions, we encoded the design contexts into three main categories and eight sub-categories (see Section IV.B).

Step 2.b. We analyzed the answers of the collected posts. Through analyzing the answers of the posts, we can identify developers' typical considerations on the design problems when they employ architecture patterns with respect to QAs and design contexts. TABLE II shows the extracted data items and the data analysis methods employed for the RQs.

To mitigate any personal bias in data collection and analysis, we first performed a pilot of data collection and analysis by randomly selecting 45 posts in order to guarantee that all the authors had a consistent understanding about the process of data collection and analysis. In the actual data collection and analysis, the first author conducted data collection and analysis, the second and third authors checked the results, and any controversial results were discussed and resolved by all the authors. Note that, we excluded the posts when the researchers judged that the posts are irrelevant or not valuable. We retrieved 3980 architecture pattern posts using the keywords related to popular architecture patterns (i.e., **Step 1.a**). Then we manually checked the candidate posts, and if the posts concern one of the QAs, we included the posts for further analysis (i.e., **Step 1.b**). Finally, 748 posts were collected for data extraction and analysis (i.e., **Step 2**). In addition, we used MAXQDA[1] as the tool to analyze the qualitative data (all the encoding and the data of this study has been made available online[2]).

TABLE II. DATA ITEMS AND THE DATA ANALYSIS METHODS USED FOR THE RESEARCH QUESTIONS

#	Data item	Data analysis method	RQ
D1	Quality attribute	Descriptive statistics	RQ1.1
D2	Architecture pattern	Descriptive statistics	RQ1.1
D3	Design context	Descriptive statistics	RQ1.2
D4	Design problem	Constant comparison	RQ2
D5	Design solution	Constant comparison	RQ3

IV. RESULTS

A. RQ1.1: Quality Attributes Considered with Architecture Patterns

We investigated this RQ from two aspects. Firstly, we explored whether developers associated some architecture patterns with certain QAs more than others. As developers might be interested in employing specific architecture patterns and focus on certain QAs, we listed the five most popular architecture patterns with their frequently concerned two QAs in Fig. 3. The results show that the frequently employed architecture patterns (i.e., MVC, Layered, SOA, Pipe & Filter, and Reflection) did not come as a surprise, as they also receive the most attention in the software architecture literatures [1]. Furthermore, for different architecture patterns, developers concern QAs differently. For example, developers are mostly concerned with Maintainability and Reusability when they employ the MVC pattern.

Besides, we identified the frequency of QAs discussed in various architecture patterns of the collected posts The importance of each QA can vary in different types of projects. Developers specified QAs using different terms. For example, "*response time*", "*capacity*", "*latency*", "*throughput*", and "*execution speed*" are related to Performance; terms like "*modifiability*" and "*stability*" are associated with Maintainability. We referred to a wordlist[3], which is specific in the software engineering field for identifying QAs, to investigate the frequency of QAs. Fig. 4 shows the popularity of eight QAs with their five most considered architecture patterns, and the most discussed QAs are Performance (184 posts) and Maintainability (123 posts), and the least discussed QAs are Portability (52 posts) and Compatibility (39 posts).

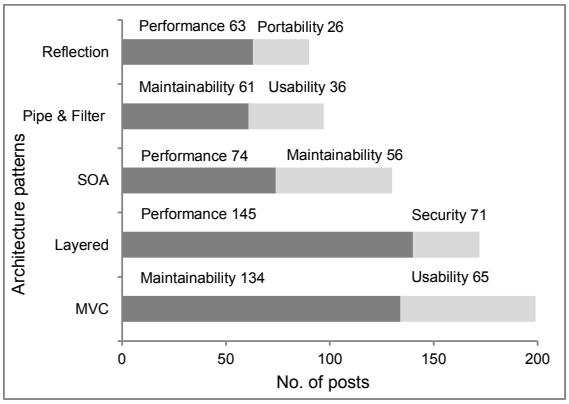

Fig. 3. Five most popular architecture patterns with their frequently concerned QAs.

B. RQ1.2: The Discussed Design Contexts in the Posts

In most of the collected posts, developers provided a brief description of the project background, and then responders provided potential solutions and rationale based on the given design problems and the design contexts. Each category of context comprises a set of subtypes. We identified and analyzed what design contexts developers discussed and were concerned with when they employed architecture patterns with respect to QAs. Each category of context comprises a set of subtypes. The classification of design contexts used in this study is adapted from the classifications in [2] and [19], and we classified design context into three categories: Application, Platform, and Organizational context (listed in TABLE III). For each category, there are more than one sub-categories. The description gives a summary of the sub-categories and the number of posts in each sub-category.

Application context comprises two subtypes: application type and application domain. Each application can be of different types (e.g., information system) [19], and this subtype is discussed most (30% posts) by developers when they introduce the backgrounds of the system. As one developer put it "*A system has its own application type, each application type has its own candidate architecture patterns*".

[1] http://www.maxqda.com
[2] https://tinyurl.com/ycsvmuhn
[3] http://softwareprocess.es/y/neil-ernst-abram-hindle-whats-in-a-name-wordlists.tar.gz

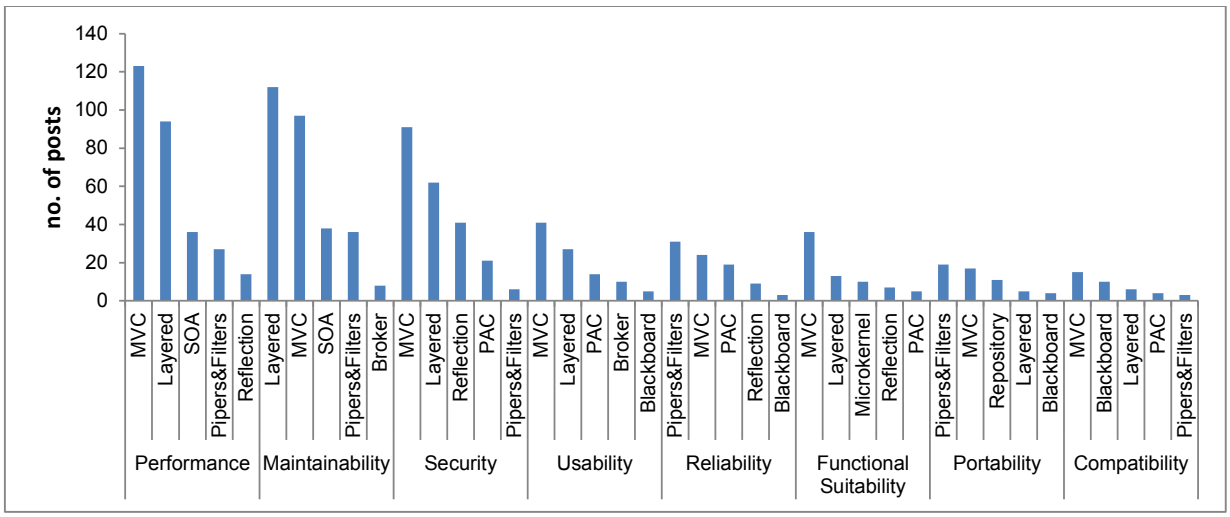

Fig. 4. The number of eight QAs discussed in the architecture pattern posts.

We extracted the most discussed application types and their frequently employed architecture patterns in TABLE IV. Application domain is another consideration that is specifically associated with QAs. For example, as one developer put it: "*Some web applications, banking websites in particular, prevent you from using … Is there a particular architectural pattern they are following to achieve this? What are their goals? How does this increase security*."

Platform context contains the technology elements that the system runs (e.g., the network capabilities) and a user employs to access an application (e.g., operating system or the installed applications). Platform context comprises three subtypes (i.e., software, hardware, and network context).

Organizational context is the least discussed design context, which comprises considerations such as how a company is organized, stakeholders, development schedule and financial factors.

TABLE III. A SUMMARY OF DESIGN CONTEXTS IN THE POSTS

Design Context	Subtype	Description	Count
Application context	Application type	The 214 posts describe the application types. They can be categorized by six types of applications, for example web application, complex system, and data flow system.	214
	Application domain	We found seven application domains amongst the 165 posts on architecture patterns.	165
Platform context	Hardware context	This subtype describes the hardware used in the system, such as laptop or mobiles devices.	185
	Software context	This subtype describes the software elements of the system, such as the operating system or installed applications.	64
	Network context	This subtype describes the network environment (such as bandwidth).	41
Organizational context	Stakeholder context	This subtype describes the stakeholders, such as managers, architects, developers, and users.	52
	Development schedule	This subtype describes development schedule, such as time-to-market.	24
	Financial factors	This subtype describes the financial constraints on development that stakeholders consider.	6

TABLE IV. APPLICATION TYPES AND THEIR FREQUENTLY EMPLOYED ARCHITECTURE PATTERENS

Application Type	Architecture Pattern
Web Application	MVC, SOA
Dataflow System	Pipe & Filter, Event-driven
Distributed System	Client Server, Broker, Peer-to-Peer

C. RQ2: Typical Design Problems

We analyzed the questions of the collected posts to identify what design problems developers seek to resolve when they employ architecture patterns with respect to QAs. In TABLE V, we listed five common design problem topics, their descriptions, and ranks by frequency.

Architecture pattern selection/comparison has the highest frequency of discussion. There are two common topics: 1) Developers ask about selecting and comparing different architecture patterns in terms of specific QAs and design contexts. As one developer asked: "*As I want to keep the application scalable, maintainable & flexible, I am thinking on which architecture I should use. In this connection I stumbled over the N-Tier and MVC Patterns?*";

2) Developers ask the difference between several architecture patterns. For example, "*What is the difference between Micro-service and Monolithic architecture patterns?*".

Characteristics of specific architecture patterns. Developers ask questions about the characteristics or explanation of unfamiliar architecture patterns. For example "*How do components interact in MVC pattern*".

Trade-offs of quality attributes are related to the considerations of balancing QAs when developers employ a specific architecture pattern. For example, one developer put it "*Now, I'm using MVC pattern for web development. Do you think it's worth trading off some performance for code-quality and maintainability*".

Technological issues. Developers ask for the detailed technological solutions for the issues. The issues can be related to various aspects, such as technology issues, the use of tools, and bugs. For example, one developer put it "*I'd like to use the Microservices architectural pattern for a new system, but I'm having trouble figuring out how to share and merge data between the services when the services are isolated from each other. In particular, we are concerned about security*".

Refactoring. Developers seek suggestions for architecture or code refactoring to improve certain QAs when employing an architecture pattern. For example, as one developer put it "*I'm trying to re-factoring ASP.NET MVC routing rules in C++ for my own MVC application. I would love for any suggestions for efficiency improvements*".

TABLE V. QUESTION TOPICS OF THE COLLECTED POSTS

Question Topic	Description	Count
Architecture pattern selection/ comparison	There is a set of architecture patterns and their variants. In this topic, developers ask for the difference between architecture patterns (in terms of certain QAs and design contexts).	231
Characteristics of specific architecture patterns	Understanding the characteristics (e.g., interaction of components) of unfamiliar architecture patterns is quite challenging. In this topic, developers ask for the information of certain architecture patterns.	194
Trade-offs of quality attributes	The quality attributes usually impact on each other. In this topic, developers ask questions about balancing various QAs when they employ specific architecture patterns.	98
Technological issues	Developers might face various issues when they employ architecture patterns with respect to QAs. In this topic, developers ask for the detailed technological solutions for the issues.	78
Refactoring	In this topic, developers focus on changing an existing body of architecture or code to improve its internal structure for improving specific QAs.	45

D. RQ3: Typical Design Considerations

The answers of the posts are the potential solutions for the design problems asked in the posts. We identified the typical considerations of the answers in three topics.

Relationships between QAs and architecture patterns. Architecture patterns selection is a multi-criteria decision-making process. Functional requirements, QAs, and design contexts are important for architecture pattern selection. In this study, about 65% posts indicate that QAs play a significant role for system design, and satisfying QAs is propounded as a key consideration in designing or selecting an appropriate architecture pattern for a system. For example, one developer put it "*extensibility and maintainability were cited as motivations for employing the MVC pattern*". The use of an architecture pattern can influence one or more QAs in different ways. We extracted the relationships between QAs and architecture patterns from the developers' viewpoints and listed the results in TABLE VI. We classify that influence as positive, neutral, or negative, in which "+" denotes that the architecture pattern benefits a specific QA, "-" shows that the architecture pattern hinders a specific QA, and "=" indicates that the relationship between an architecture pattern and a specific QA is neutral. We provide the numbers of posts regarding the relationships of "positive", "negative", or "neutral" between architecture patterns and QAs in each cell in TABLE VI. Note that, if developers do not make a point explicitly whether a specific QA is benefited or hindered by the architecture patterns (e.g., the relationship between the Layered pattern and Functional Suitability), we used "N/A" to denote the relationships.

How design contexts impact QAs. Design contexts and QA requirements can influence each other [14]. For example, application domains heavily influence QA considerations, and we identified seven most discussed application domains with their important QAs in TABLE VII. Note that there may be other important QAs in various application domains, but we only extracted and listed the QAs and the application domains based on the selected posts of this study. In addition, approximately 40% posts indicate that QAs are influenced by design contexts. Various contextual factors influence QAs. For example, QAs issues can be caused by missing and ambiguous platform contextual elements. As developers put it: "*I definitely do value my own time over application performance on the server side. If I notice that my site is not performing well enough, and upgrading the server hardware is an alternative solution that could solve my problem with looking at the code".* Organizational contextual elements can also impact QAs, and as one developer put it: "*It has been my recent experience on big government projects that quality attributes often suffers due in part to schedule constraints*".

Trade-offs between QAs in specific architecture patterns. When an architecture pattern is selected in design,

TABLE VI. RELATIONSHIPS BETWEEN QAS AND ARCHITECTURE PATTERNS EXTRACTED FROM THE POSTS

	QA							
Architecture Pattern	Maintainability	Reliability	Performance	Portability	Usability	Compatibility	Functional Suitability	Security
MVC	+ (35) = (4)	= (12)	= (3)	= (3)	+ (31)	- (3)	N/A	= (9)
Layered	+ (30)	+ (21)	- (14) = (2)	+ (5)	N/A	- (2)	N/A	+ (29) = (1)
Pipe & Filter	+ (15) = (2)	- (5) = (1)	+ (2) - (1)	= (3)	- (10) = (1)	+ (3)	N/A	N/A
Repository	- (2)	- (3)	N/A	+ (2) = (1)	= (6)	+ (5)	N/A	= (3)
Blackboard	- (3)	= (1)	+ (5)	= (2)	= (3)	= (1)	N/A	= (2)
Broker	+ (5)	+ (2)	N/A	+ (2)	+ (1)	N/A	= (4)	+ (12) = (1)
PAC	+ (3)	= (1)	- (2)	+ (1)	+ (2)	N/A	N/A	= (1)
SOA	+ (27)	= (1)	= (2)	N/A	= (2)	N/A	N/A	= (2)
Client server	- (2)	- (3)	N/A	+ (1)	+ (1)	N/A	N/A	+ (2)
Batch Sequential	= (1)	+ (3)	N/A	= (2)	= (1)	N/A	N/A	N/A

TABLE VII. APPLICATION DOMAINS AND THEIR IMPORTANT QAS

Application Domain	**Important QAs**
Banking system	Security, Maintainability,
Education system	Usability
Game system	Performance, Efficiency
Healthcare system	Performance, Usability
Trading system	Performance, Security
Railway system	Performance, Maintainability, Security
Manufacturing system	Usability

trade-offs among multiple forces are often part of the consideration [23]. One software system may not be able to satisfy all desired QAs at the same time. QAs can impact on one another both positively and negatively [30]. Some QAs can improve through using specific architecture pattern, whilst others can worsen. There were some explicit discussions about the tradeoffs between QAs in specific architecture patterns. For example, in order to improve the performance of a system, one proposed solution is to use cross-layer dependencies in the Layered pattern, which might compromise system modularity, and further lead to the deterioration of reusability. This is an erosion of the Layered pattern. When developers determine the priorities of QAs, trade-offs between QAs are made accordingly.

V. DISCUSSION

Whilst architecture patterns are well known and they are often accompanied by certain design guidelines, there is little knowledge on how architecture patterns are used in certain design contexts. Some of the issues lie in the lack of knowledge in their relationships with QAs and the contexts. In this section, we discuss the implications of the results from Section IV. Particularly, how do developers consider quality attributes, design contexts, and architecture patterns and their inter-relationships in design?

Quality attributes are interrelated and to design elements. The discussion and negotiation of QAs is an iteration process (e.g., throughout the software system development lifecycle) that can shape the design decisions. Designing for QAs cannot be considered in isolation, and we summarized a set of inter-related knowledge (from RQ2 and RQ3) that influences the design considerations for QAs. A relationship diagram (see Fig. 5) helps developers consider their specific situations. Firstly, developers identify the priorities of QAs that are of concern, like the knowledge about particular types of QAs and their associated subtypes and terminology. Priorities of QA requirements can arise from considerations of a set of factors. For example, important QA requirements that are largely dependent on the functional requirements and design contexts of a particular system, (e.g., banking system considers security as vital). Furthermore, developers consider design decisions (such as selection of architecture patterns) based on the vital QAs. Finally, QAs are interrelated. Some QAs are identified as critical, dominant and are vital to the success of the system. Trade-offs occur because when a decision positively impacts a QA, it may negatively affect other QAs.

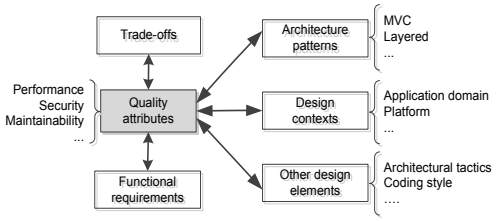

Fig. 5. The typical design elements for QAs consideration.

Importance of design contexts: The results from RQ1 and RQ2 show that design contexts are often omitted by developers. For example, in the "Technological issue" topic (see TABLE V) developers prefer to seek technological design solutions rather than consider the influence of design context to QAs. One reason could be that design contexts are conditions that can influence design decisions but they are not specified explicitly as requirements. In addition, stakeholders do not make contextual knowledge

explicit and document them in a systematic way. The missing and underappreciated (not-well managed) contextual knowledge will lead to inadequate considerations for the design problems. As one developer stated "*the background and requirements of your project is VERY important, but you don't mention*". Moreover, we conclude that design context is not only important for QAs, but also vital for software design. One potential reason could be that design contexts comprise the knowledge spanning the whole development lifecycle (see TABLE III). Furthermore, design contexts are related to requirements, design decisions, and risks. Developers need to be aware of them when they make design decisions. This could motivate researchers to spend extra efforts on contextual knowledge management, and further research regarding the integration of contextual knowledge management into existing software development is required.

Analysis of architecture pattern usage in practice: From TABLE V in Section IV.C, we see that the most frequently asked design pattern problems are the selection and comparison of architecture patterns and the characteristic of using specific architecture patterns. To know how to use an architecture pattern, a developer needs to understand some intricate details of the architecture pattern. But there are few guidelines on the use of architecture patterns with respect to the impacts on QAs and architectural design. Thus, there is a need to guide developers based on the problems they ask. As such, we listed three considerations (see TABLE VIII) for developers on architecture pattern usage (e.g., architecture selection) in practice.

TABLE VIII. THREE CONSIDERATIONS ON ARCHITECTURE PATTERN USAGE

	Considerations		
	QA drive	**Characteristics of application types**	**Benefits and challenges**
Architecture pattern usage	Relationships between architecture patterns and various QAs (about 65% posts) - see TABLE VI	Patterns typically used for certain applications with specific QA requirements (about 30% posts) - see TABLE IV	Be aware of the benefits and challenges for architecture pattern usage (about 15% posts)

As illustrated in Section IV.D, the relationships between QAs and architecture patterns also drive the use of architecture patterns. As one developer put it "*Focus on the non-functional requirements, and from there try to pick an architectural pattern. A software quality analysis will be helpful*". About 30% developers in this study associated architecture pattern with application types. As one developer put it "*The Broker architectural pattern can be used to structure distributed software systems with decoupled components that interact by remote service invocations. A broker component is responsible for coordinating communication, such as forwarding requests, as well as for transmitting results and exceptions*". We see that certain patterns are popular in certain application types. They are listed in TABLE IV. Whilst it is well understood that the use of architecture patterns brings certain benefits such as design reuse, code reuse, and it benefit for developers to understand a system's structure [1], we suggest that realizing these benefits would depend on the right design contexts. For example, one developer put it "*in order for architects and developers to make design decisions correctly and efficiently, they need information about how architecture patterns and other decisions (e.g., architecture tactics) interact, but it was not easy task*".

VI. THREATS TO VALIDITY

The threats to the validity of this study are discussed in this section according to the guidelines in [24].

Construct validity in this study focuses on whether the way we collect and analyze the data from Stack Overflow can answer the RQs. A threat is our study is an incomplete search terms may omit some relevant architecture patterns posts. To mitigate this threat, we checked the various relevant literatures and chose the terms for gathering candidate architecture posts, and we used ISO 25010 standard to filter the posts. The other threat is the applying ISO 25010. We have no evidence that ISO 25010 is the most correct and detailed standard and developers may be ignorant of it, but it presently constitutes the most widely used software quality model. In addition, there is a risk that the results of this study might be affected by the interpretation of the posts collection and analysis by different researchers. To overcome this threat, we conducted a pilot data collection and analysis (using 45 posts), any controversial results for data collection and analysis were discussed and resolved among all the authors.

Internal validity focuses on the factors that may have an influence on the validity of the results. We employed a descriptive statistics method to present the results of this study, and this threat is minimized. In addition, the first author used MAXQDA for data labelling and encoding, and the second and third author reviewed the labelling and encoding results. This can partially reduce the threat of bias in the qualitative data analysis.

External validity refers to the degree to which the findings of the study can be generalized. To mitigate the threat that the data could be unrepresentative of the RQs we investigate, we need a good source of developers' forum. We chose one of the top Q&A sites (i.e., Stack Overflow), which is large, widely used, and popular with tens of thousands of developers worldwide.

Reliability focuses on whether the study yields the same results if other researchers replicate this study, which is related to the data collection and analysis in Stack Overflow. By making explicit the process of data collection and analysis of this study, and the data online, we believe that this study can be replicated.

VII. CONCLUSIONS AND FUTURE WORK

In this study, we collected 748 posts from Stack Overflow. We analyzed how developers consider QAs, design contexts, and architecture patterns when designing using DAT as our conceptual model. We have provided empirical evidences to confirm that design contexts and QAs are important aspects to consider when applying architecture patterns. Developers often need to know relationships such as how architecture patterns influence QAs and under what circumstances. With these relationships, we have listed three considerations to help developers select architecture patterns with respect to QAs and design context. These considerations highlight the trade-offs between QAs in different applications and design contexts. Our next steps include: (1) investigating the relationships between QAs and other architectural design elements (e.g., architecture tactics) by using more data source (e.g., issue tracing system of open source software projects). (2) employing semi-automatic approaches to extract QA requirements and construct traceability links between QAs and design decisions.

REFERENCES

[1] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 3rd Edition, Addison-Wesley Professional, 2012

[2] A. Bedjeti, P. Lago, G. A. Lewis, R. C. de Boer, and R. Hilliard, "Modeling context with an architecture viewpoint". In: Proceedings of the 1st IEEE International Conference on Software Architecture (ICSA), Gothenburg, Sweden, 2017, pp. 117-120.

[3] D. Gross and E. Yu, "From non-functional requirements to design through patterns". Requirements Engineering, 2001, vol. 6, no.1, pp. 18-36.

[4] G. Me, G. Procaccianti, and P. Lago, "Challenges on the relationship between architectural patterns and quality attributes". In: Proceedings of the 1st IEEE International Conference on Software Architecture (ICSA), Gothenburg, Sweden, 2017, pp. 141-144.

[5] J. Vassileva, "Toward social learning environments". IEEE Transactions on Learning Technologies, 2008, vol. 1, no. 4, pp. 199-214.

[6] N. B. Harrison and P. Avgeriou, "Analysis of architecture pattern usage in legacy system architecture documentation". In: Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), Vancouver, BC, Canada, 2008, pp. 147-156.

[7] P. Velasco-Elizondo, R. Marín-Piña, S. Vazquez-Reyes, A. Mora-Soto, and J. Mejia, "Knowledge representation and information extraction for analysing architectural patterns". Science of Computer Programming, 2016, vol. 121, pp. 176-189.

[8] J. Zou, L. Xu, M. Yang, X. Zhang, and A. Yang, "Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis". Information and Software Technology, 2017, vol. 84, pp. 19-32.

[9] ISO/IEC, ISO/IEC 25010:2011, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

[10] B. Vasilescu, A. Capiluppi, and A. Serebrenik, "Gender, representation and online participation: A quantitative study of stackoverflow". Interacting with Computers, 2013. vol. 26, no. 5, pp. 488-511.

[11] G. Me, C. Calero, and P. Lago, "A long way to quality-driven pattern-based architecting". In: Proceedings of the 10th European Conference on Software Architecture (ECSA), Copenhagen, Denmark, 2016, pp. 39-54.

[12] N. B. Harrison and P. Avgeriou, "Leveraging architecture patterns to satisfy quality attributes". In: Proceedings of the 1st European Conference on Software Architecture (ECSA), Madrid, Spain, 2007, pp. 263-270.

[13] K. E. Harper and J. Zheng, "Exploring software architecture context". In: Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture (WICSA), Montreal, QC, Canada, 2015, pp. 123-126.

[14] A. Tang and M. F. Lau, "Software architecture review by association". Journal of Systems and Software, 2014, vol. 88, no.2, pp. 87-101.

[15] D. Ameller, M. Galster, P. Avgeriou, and X. Franch, "A survey on quality attributes in service-based systems". Software Quality Journal, 2016, vol. 24, no. 2, pp. 271-299.

[16] S. Kim, D. K. Kim, L. Lu, and S. Park, "Quality-driven architecture development using architectural tactics". Journal of Systems and Software, 2009, vol. 82, no. 8, pp. 1211-1231.

[17] M. Soliman, M. Galster, A. R. Salama, and M. Riebisch, "Architectural knowledge for technology decisions in developer communities: An exploratory study with stackoverflow". In: Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), Venice, Italy, 2016, pp. 128-133.

[18] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? An analysis of topics and trends in stack overflow". Empirical Software Engineering, 2014, vol. 19, no. 3, pp. 619-654.

[19] K. Petersen and C. Wohlin, "Context in industrial software engineering research". In: Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM), Lake Buena Vista, Florida, USA, 2009, pp. 401-404.

[20] B. G. Glaser and A. L. Strauss, The Discovery of Grounder Theory: Strategies for Qualitative Research. Aldine Publishing, New York, 1968.

[21] E. Golden, B. E. John, and L. Bass, "The value of a usability-supporting architectural pattern in software architecture design: A controlled experiment". In: Proceedings of the 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, USA, 2005, pp. 460-469.

[22] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions". In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, USA, 2005, pp. 109-120.

[23] M. Salama, R. Bahsoon, and N. Bencomo, "Managing Trade-offs in Self-adaptive Software Architectures: A Systematic Mapping Study". Managing Trade-offs in Adaptable Software Architectures, Chapter 11, 2007, pp. 249-297.

[24] M. Höst, P. Runeson, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in Software Engineering. Springer, 2012.

[25] D. Pagano and W. Maalej, "How do open source communities blog?". Empirical Software Engineering, 2013, vol. 18, no. 6, pp. 1090-1124.

[26] ISO/IEC/IEEE. ISO/IEC/IEEE Std 42010:2011. Systems and Software Engineering - Architecture description, 2011.

[27] A. Sharma, M. Kumar, and S. A. Agarwal, "Complete survey on software architectural styles and patterns". Procedia Computer Science, 2015, vol. 70, pp. 16-28.

[28] M. A. Babar, "Scenarios, quality attributes, and patterns: Capturing and using their synergistic relationships for product line architectures". In: Proceedings of the 11th Asia-Pacific Conference on Software Engineering (APSEC), Busan, Korea, 2004, pp. 574-578.

[29] S. Easterbrook, J. Singer, M. A. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research". Guide to Advanced Empirical Software Engineering, Chapter 11, Springer, 2008, pp. 285-311.

[30] K. Henningsson and C. Wohlin, "Understanding the relations between software quality attributes - A survey approach". In: Proceedings of the 12th International Conference for Software Quality (ICSQ), Ottawa, Canada, 2002, pp. 1-13.