Improving the Search for Architecture Knowledge in Online Developer Communities

Mohamed Soliman*, Amr Rekaby Salama*, Matthias Galster†, Olaf Zimmermann‡, and Matthias Riebisch*
*Department of Informatics, University of Hamburg, Germany
Email: {soliman, salama, riebisch}@informatik.uni-hamburg.de
† University of Canterbury, Christchurch, New Zealand
Email: mgalster@ieee.org
‡ University of Applied Sciences of Eastern Switzerland (HSR FHO), Rapperswil, Switzerland
Email: olaf.zimmermann@hsr.ch

***Abstract*—To make good architecture design decisions, developers need to know about suitable architecture solution alternatives. However, with the rapid and continuous increase of solution alternatives (e.g. technologies, patterns, tactics) it is challenging to acquire architecture knowledge and to ensure that this knowledge is up to date. Our goal is to improve how architects search for architecturally relevant information in online developer communities. We developed a new search approach for architecturally relevant information using Stack Overflow as an example of an online developer community. Our search approach differs from a conventional keyword-based search in that it considers semantic information of architecturally relevant concepts in Stack Overflow. We also implemented the search approach as a web-based search engine. To show the effectiveness of the search approach compared to a conventional keyword-based search, we conducted an experiment with 16 practitioners. To ensure realism of the experiment, tasks given to practitioners are based on real scenarios identified in a separate interview study with a different set of practitioners. The experiment showed that the new search approach outperforms a conventional keyword-based search.**

***Keywords*-software architecture knowledge; search approach; online developer communities; Stack Overflow**

I. Introduction

A. Problem and Context

The number of architectural solutions to design problems (e.g. technologies, patterns, tactics) has increased significantly over the last decade [1]. Moreover, solutions evolve constantly as the half-life of software engineering knowledge, tools and technologies is about five years [2]. This makes it challenging to learn, explore and to keep up to date with solution alternatives. To support architects when coping with the fast change of architecture solution alternatives, researchers have proposed architecture knowledge (AK) repositories (e.g. [3], [4]). These repositories contain knowledge about technologies, patterns, tactics etc. and their characteristics (e.g. how a technology helps implement a certain architecture pattern). Repositories allow architects to browse and learn about new solutions. However, repositories rely on *manually* capturing and curating AK [5]. This raises practical problems, because knowledge and experience about technologies tends to be shared by architects and developers through different knowledge sharing channels, such as online developer community web pages (e.g. blogs, forums) and vendor websites. Repositories therefore need to aggregate information from different sources [5]. Also, as argued above, AK evolves fast, so manually captured knowledge becomes out of date quickly.

On the other hand, online developer communities (such as Stack Overflow) encourage individuals to share their experience about software engineering problems continuously and frequently. This is because such communities reward submitting valuable questions and answers with increasing reputation. However, while online developer communities capture large amounts of information in posts (i.e. questions and answers), this information is represented as unstructured text. Moreover, the abstract nature of architectural concepts makes it difficult for keyword-based searches to find architecturally relevant information: The same architecture concept could be presented with many keywords, and the same keyword might refer to different concepts. For example, a keyword-based search for "Apache webserver performance" will not find posts which contain the "throughput", even though throughput is related to performance and could therefore be relevant. Also, some words could occur in different posts with different meanings. For example, "server" could relate to architecture components in an architecture-relevant post, but refer to a deployment environment in programming-related posts.

B. Paper Goal and Contributions

According to Kazman and Cervantes, finding architectural solutions typically follows three iterative steps [6]:

1) *Identify design concepts*: Types of architectural solutions (e.g. patterns, tactics, technologies) and candidate solutions (e.g. layers in case of patterns, RabbitMQ for broker technologies) are identified.
2) *Select design concepts*: Based on the benefits and drawbacks of alternative design concepts, the most appropriate solutions are selected.
3) *Instantiate architecture elements*: Selected design concepts are customized for the given design issue (e.g. for a layered pattern, we need to decide on the number of layers, components in layers and their communication).

When performing these steps, developers search various sources, including online communities, e.g. Stack Over-

flow [7]. Even though developer communities typically help developers solve coding problems, they also provide architecturally relevant information [8], [9]. Furthermore, online developer communities offer free information, fast responses to questions and diverse solutions and opinions. Therefore, we aim at answering the following research question:

> **RQ:** *How can we improve the search for architecturally relevant information in online developer communities?*

This research question is motivated by the shortcomings of traditional keyword-based search approaches which cannot deal with the ambiguity of terms for architectural concepts. Gorton et al. [5] argue that it is challenging for practitioners to create effective search queries for relevant architecture information and internet search engines return many irrelevant results. Thus, we need domain-specific search approaches [7].

To answer this RQ, we developed an enhanced search approach to search for architecturally relevant information in Stack Overflow. We selected Stack Overflow since it is currently the biggest[1] and most popular online developer community (see [7]) and does indeed contain architecturally relevant information [8], [9]. Also, Gorton et al. recommend Stack Overflow for solving architectural issues [5]. Our **contributions** are the following:

- We developed a new search approach to help find architecturally relevant information on Stack Overflow. The search approach (see Section III) utilizes the classification approach to filter and re-rank Stack Overflow posts based on their suitability for architecture design activities.
- We developed and evaluated classification approaches to automatically separate architecture-relevant posts from pure programming-related posts in Stack Overflow. The classification approaches (see Section IV) also classify architecture-relevant posts into sub-categories to support specific architecture design activities.
- We performed an experiment with 16 practitioners to show the effectiveness of the search approach for supporting architecture design tasks. The new search approach outperforms a conventional keyword-based search. The evaluation is presented in Section V.

Our work utilizes semantic information captured in an ontology of AK concepts, rather than depending solely on lexical features of Stack Overflow posts. In summary, our work shows the feasibility of building specialized search engines for architectural information in online developer communities.

Before presenting our contributions in Sections III, IV, and V, we discuss some of the conceptual foundations of our work in Section II. We then discuss our findings (including future work) in Section VI, present related work in Section VII and conclude the paper in Section VIII.

[1]As of March 14, 2018, stackoverflow.com has 15,508,519 posts (12,812 posts have the "architecture" tag). Other communities have fewer posts. For example, as of Mach 14, 2018, softwareengineering.stackexchange.com has 48,662 posts (2,102 posts have the "architecture" tag.)

II. Background

A. Architecture-Relevant Posts (ARPs)

Soliman et al. [8] define ARPs in Stack Overflow as "posts with questions related to performing an architecture design activity. The questions in ARPs sometimes consider quality attributes and contextual factors, and the answers involve experience and knowledge about technology solutions, their differences and capabilities". Below is a Stack Overflow post[2] that discusses the components design when using technologies "RIA Services and Entity Framework". The question is about assigning responsibility for the business logic. A reply describes a possible component design.

> ***Question:*** *"I have been experimenting recently with Silverlight, RIA Services, and Entity Framework using .NET 4.0. I'm trying to figure out if that stack makes sense for use in any of my upcoming projects (...) but I'm struggling to decide how an application on top of this stack should be architected. (...) So, my questions: What is the best location for business logic (rules, validations, behaviors, authorization) in an application using this stack? ".* ***Answer:*** *" We've decided to build a domain model on top of the RIA entities. Additionally, we elected to follow the MVVM pattern to model UI interactions. So far, I've noticed the following benefits: 1. Domain classes are a nice place to put business logic including complex validations. 2. Domain classes use the RIA entities and context as interface to data store. 3. Domain classes are modeled after business concerns."*

Soliman et al. [8] also categorized ARPs based on the purpose of posts and types of architectural solutions discussed:

- *Technology identification posts* concerned with searching for suitable technologies for a design problem.
- *Technology evaluation posts* concerned with evaluating one or more technologies with regards to certain aspects (e.g. performance).
- *Features and configuration posts* describing features of a technology solution or the configuration (component design) of an architecture (e.g. the example given above).

Categorizing ARPs allows grouping ARPs, which contain similar AK concepts. For example, *Technology evaluation posts* contain benefits and drawbacks about technologies, which is useful when performing the *Select design concepts* design step. Our search approach relies on providing suitable types of ARPs for each design step (see Section III-B).

B. Architecture Knowledge Ontology in Stack Overflow

Some of the steps of the enhanced search approach in Section III-A rely on an ontology for architectural concepts. Soliman et al. [9] proposed an ontology to specify the concepts relevant to software architecture in online developer communities. We provide a brief description for some ontology classes (i.e. "explicit specifications of a concept" [10]). A complete

[2]https://stackoverflow.com/questions/2897513

TABLE I
EXAMPLES OF COMPOSITE ONTOLOGY CLASSES

ID	Ontology class	Example post
CONF	Architecture Configuration	Push data from server to client
REQ	Requirement and Constraint	We need reliable delivery
FEAT	Technology Feature	ProtoBuffers offer serialization
ASTA	Technology Benefit or Drawback	MSMQ does not provide messaging patterns
ADD	Recommended Decision	WCF seems best suited
DR	Decision Rule	If you want to make API public do it in RESTful way

description is provided by Soliman et al. [9]. For example, the sentence *"If performance is your main criteria, you should definitely look at ZeroMQ."* in an ARP[3] contains the following ontology classes:

1) *Simple* ontology classes: These classes are represented in text as *single words* which refer to certain architectural concepts. For example, the words "ZeroMQ" and "performance" are two simple concepts assigned to ontology classes *Technology Solution (TEC)* ("ZeroMQ") and *Quality Attribute (QA)* ("performance").
2) *Lexical triggers*: They indicate individual words that are not specifically related to an architectural concept, but still capture important meanings for the whole sentence. For example, in the above statement, words like "criteria" would be of ontology class *Concern Noun*.
3) *Composite* ontology classes: These are represented in text as *clauses* and *sentences* and are composed of simple ontology classes and lexical triggers to capture semantics. For example, "performance is your main criteria" would be of ontology class *Requirement and Constraint (REQ)*, "you should definitely look at ZeroMQ" would be of ontology class *Recommended Decision (ADD)* and the whole statement above would be of ontology class *Decision Rule (DR)*. Table I lists additional examples of composite ontology classes.

III. SEARCH APPROACH FOR ARCHITECTURALLY RELEVANT INFORMATION

A. Overview of Enhanced Search Approach

The most common way to search online developer pages such as Stack Overflow are web-based search approaches [7]. Therefore, our search approach also utilizes ideas from keyword-based web search approaches. Our approach enhances a keyword-based search using two additional activities which complement a pure keyword-based search with semantic information about architecture relevance of posts: A) Filtering and separating "architecture-relevant" posts from other types of posts. B) Re-ranking "architecture-relevant" posts based on their significance to support a certain architecture design step. Fig. 1 shows the proposed search approach. Step 3 and Step 4 are our main contributions, which differentiate our enhanced search from a conventional keyword-based search:

Step 1 – Specify search query + design step: Similar to a normal keyword-based search, a user defines a query with keywords relevant to a design problem. However, different to a conventional keyword-based search, a user also specifies the current design step (used in Step 4 to filter and re-rank posts). Here, we utilize the three design steps proposed by Kazman and Cervantes [6], see Section I-B (identify design concepts, select design concepts, and instantiate architecture elements).

[3]stackoverflow.com/questions/17806977

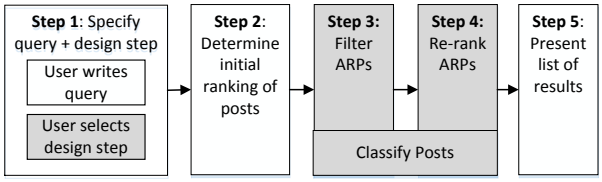

Fig. 1. Enhanced search process (gray elements are the extensions of a keyword-based search)

Step 2 – Determine initial ranking of posts: This step is the same as in conventional keyword-based search approaches. It first parses, stores and indexes all words in posts that are searched to facilitate fast retrieval through mapping between posts and words (note that our implementation for this step uses existing libraries to implement indexing, see Section III-C). Then, this step assigns weights to these words based on their ability to differentiate posts (e.g. using the TF/IDF weighting scheme [11]). Finally, based on the keywords in a search query, a similarity score between keywords and the words in posts (considering the weight of each word) is calculated for each post to assess its relevance to the query.

Step 3 – Filter architecture-relevant posts: This step requires information about the type of post, i.e. is a post an ARP or not, and if so, what type of ARP (see types of posts in Section II-A). We obtain this information by classifying posts. Details about the classification are provided in Section IV. All architecture-relevant posts with a similarity score higher than zero (as computed in Step 2) are moved above programming posts in the list of search results. ARPs are sorted in an ascending order based on their similarity scores.

Step 4 – Re-rank architecture-relevant posts: Filtered posts are re-ranked according to their suitability to support the design step specified in Step 1. To relate types of ARPs to design steps, we assigned a "quota" (i.e. estimated percentages) for how types of ARPs should appear in the search results depending on the selected design step. Post types with higher quota have a bigger chance of appearing in the top results. For example, for a query for design step "identify design concepts", the top ten search results will contain 50% of "technology identification" posts, 30% "technology evaluation" posts and 20% of "features and configuration" posts. Posts with the highest similarity scores (from Step 2) are selected as part of the quota for each ARP type. Percentages for "select design concept" and "instantiating architecture elements" are 30%, 50%, 20%, and 10%, 15%, 75%, respectively. These quotas are estimated according to an exploratory study (see Section III-B).

B. Types of Architecture-relevant Posts and Design Steps

To determine the most useful types of architecture-relevant posts to support the three generic design steps proposed by

Kazman and Cervantes [6] (see Section I-B), we performed an exploratory study as follows:

1) **Link ontology classes to design steps:** Some ontology classes (see Section II-B) are more useful than others for certain design steps. Based on the definitions of ontology classes and design steps, we decided what ontology classes are more useful for which design step: For design step "*identify design concepts*", it is useful to reuse design decisions related to technology solutions. Therefore, ontology class *ADD*, which contains ontology class *TEC* has been selected as being relevant to this design step. For design step "*select design concepts*", ontology classes *QA*, *ASTA* and *DR* are important to evaluate and select solutions [12]. For design step "*instantiate architecture elements*", design decisions on architectural configurations (components and connectors) and technology features are useful to be reused [6]. Therefore, ontology class *ADD*, which contain ontology classes *FEAT* and *CONF* have been selected as being relevant to this design step.

2) **Calculate occurrences of ontology classes in posts:** Soliman et al. [9] provide a corpus of 105 ARPs classified into the three types. The posts of this corpus are annotated with more than 3,800 annotations. Each annotation represents an ontology class as defined in the ontology of Soliman et al. [9] (see Section II-A). We counted the number of occurrences of each ontology class for each type of ARP. The supplementary material[5] provides detailed statistics about the number of occurrences of annotations for each ontology class.

3) **Obtain quotas for post types:** Based on occurrences of ontology classes in the types of posts and the relevance of ontology classes for design steps, we determined quotas for types of posts and how types of posts appear in search results. For example, annotations about ontology classes *ASTA* and *DR* (important for design step "select design concepts") appear in "technology evaluation" posts 1.6 times more frequently than in "technology identification" posts, and 2.5 times more frequently than in "features and configuration" posts. Thus, for design step "select design concepts" quotas are 30% "technology identification" posts, 50% "technology evaluation" posts and 20% "features and configuration" posts. We estimated quotas for the other two design steps accordingly.

C. Implementation of Search Approach

We implemented the search approach as a proof-of-concept web-based search engine using Apache Lucene. We used Lucene, because it has successfully been applied in previous software engineering research [13]. Filtering and re-ranking of posts were built on top of the keyword-based search in Lucene. Our implementation includes its own database of posts. This database has been filled by querying Stack Overflow's API. Therefore, new posts can easily be added. Similar to conventional search engines, a list of re-ranked posts is presented to the user.

IV. Identification and Classification of Architecture-Relevant Posts

A. Overview

The search approach proposed in Section III requires filtering and re-ranking of posts based on the type of a post. Therefore, in this section we develop and evaluate an approach to identify and classify architecture-relevant posts in Stack Overflow. The goal is to classify posts into *predefined* categories (three types of ARPs, see Section II-A, and programming posts). The classification approaches that we developed and compared use different methods for *preprocessing* posts. Preprocessing transforms words in posts into a feature vector (i.e. a vector of numbers which represents important classification features). Natural language has a lexical and a semantic dimension [14]. Therefore, we explored two preprocessing methods with different *classification features*: 1) *Lexical* features of text in posts using the *Bag-of-Words* method [15] (Section IV-B). 2) *Semantic* features of text in posts using an *ontology-based* classification which captures semantics [16] (Section IV-C).

The different preprocessing methods produce feature vectors, which are used by classification algorithms (e.g. Bayesian Network) to develop a classification model. The model is developed through training previously classified posts. In Section IV-E, we present results for the best performing classification algorithms. We also experimented with an *ensemble learning* approach [17] to combine the results of different classification approaches into a single approach (Section IV-D).

B. Bag-of-Words Classification

Bag-of-Words uses individual words in posts to determine characteristics of the text [18]. The frequency of each word as well as the frequency of sequences of words in posts are used as features for training classification algorithms. To transform textual posts into a feature vector, we first removed stop words from posts[4]. This reduces noise and increases the chance for distinctive sequence of words to be transformed into single features. Then, we used an *n*-gram sequence classification approach [19] to capture the sequence of common words. The frequency of each unique sequence of words is treated as a feature. We experimented the Bag-of-Words preprocessing with several classification algorithms. The best performing algorithms are presented in Section IV-E.

C. Ontology-Based Classification

The classification approaches developed in this section use an ontology-based document classification approach [16]. To explore both the impact of individual ontology classes and sequences of ontology classes on identifying architecture-relevant posts, we separately explored single-ontology-class and multi-ontology-class classification approaches.

[4]http://astellar.com/2011/12/stopwords-for-sphinx-search/

1) Single-ontology-class Classification: The single-ontology-class approach relies on features from *separate* simple ontology classes and lexical triggers (see Section II-B). We counted the total number of words as well as the distinct number (without duplicates) of words in a post which belong to a simple ontology class or lexical trigger. Additionally, we considered commonly used features for classifying Stack Overflow posts such as the availability of source code, and the number of words and paragraphs in a post.

2) Multi-ontology-class Classification: The single-ontology-class classification does not identify composite ontology classes. Therefore, we also explore multi-ontology-class classification, which treats each possible sequence of single ontology classes and lexical triggers as features. To transform text in posts into features (i.e. sequences of ontology classes), we followed three steps: 1) *Lemmatization* converts the inflected forms of a word into a single form. This allows capturing more words for one ontology class. 2) *Abstraction* replaces words which belong to one of the ontology classes with the name of that ontology class. For example, all different technology names (e.g. RabbitMQ, MSMQ) are replaced with the TEC ontology class name. 3) *Sequential feature encoding* captures all possible combinations of sequences of ontology classes in a post via *n*-gram processing. The frequency of each unique sequence of ontology classes is treated as feature.

Similar to the Bag-of-Words classification, we experimented with several classification algorithms. The best performing algorithms are presented in Section IV-E.

D. Ensemble Learning

Combining the different approaches together would make use of the benefits of each to potentially improve the quality of classification. To achieve this, we performed ensemble learning using the *voting algorithm* [20]. The voting algorithm takes the probabilities of a post for being of a certain type as computed by each classification approach and calculates an average probability. We chose the voting algorithm after a comparison with the *boosting ensemble learning algorithm* [21]. The boosting ensemble learning algorithm did not improve the quality of the classification above the quality of the individual classification approaches combined in the ensemble.

E. Evaluation of Classification Approaches

We conducted experiments using a corpus of categorized Stack Overflow posts from Soliman et al. [8]. The corpus consists of 1,653 programming posts and 858 ARPs. The 858 ARPs are further categorized into 282 technology identification ARPs, 291 technology evaluation ARPs and 285 features and configuration ARPs. We performed a 10-fold cross-validation [22]: All posts in the corpus were randomly divided into ten equal groups. We then performed 10 experiments where we trained the classification algorithms on nine groups and tested them on the remaining group. For each of the ten experiments we calculated precision *P*, recall *R*, and *F1* as the harmonic mean of precision and recall. We used Weka [23] to conduct the experiments.

TABLE II
EVALUATION RESULTS OF CLASSIFICATION APPROACHES. P, R, AND F1 ARE AVERAGE PRECISION, RECALL AND F-SCORES

	Bag-of-words BN5G - A1	**Single-ontology LMT - A2**	**Multi-ontology VB5G - A3**	**Ensemble Learning Voting**
P	0.73	0.725	0.701	0.737
R	0.715	0.756	0.652	0.733
F1	0.722	0.725	0.672	0.734

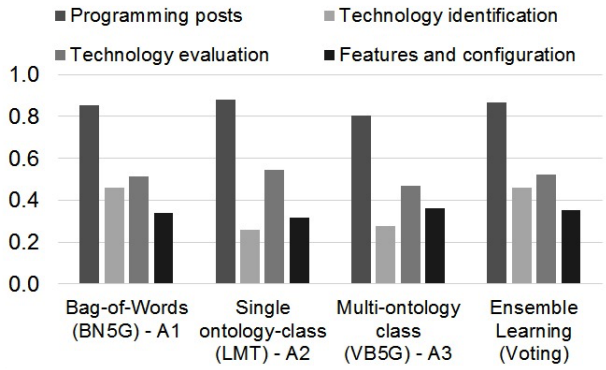

Fig. 2. Comparing *F*-scores across different classification approaches and types of posts

As mentioned in Sections IV-B and IV-C, we experimented with several classification algorithms in combination with the three preprocessing approaches (Bag-of-Words, Single-ontology, Muli-ontology). We selected classification algorithms which are known for their ability to classify documents and which have been used previously to solve software engineering problems (e.g. [24]). In this section, we present the results for the best performing classification approaches. The full results for all algorithms and experiments are available in the supplementary material[5]. The best performing classification approaches are **A1**: Bag-of-Words using Bayesian Network (BN5G) [25]; **A2**: Single-ontology-class classification [16] using logistic model trees (LMT) [26]; **A3**: Multi-ontology-class classification using Naive Bayes (VB5G) [27].

The three best performing classification approaches are combined using the voting ensemble learning. Table II presents the results of evaluation for the classification approaches (based on the average *P*, *R* and *F1* across the ten experiments for each classification approach). The results show no significant differences between the the classification approaches. However, the ensemble learning shows a slightly higher classification accuracy over the individual approaches.

The results in Table II are the average of *F1* among the four types of posts (programming posts, technology identification, technology evaluation, and features and configuration). To explore the ability of the classification approach to identify each type of posts, we computed *F1* for each type of post separately. Fig. 2 shows a comparison between the different types across the different classification approaches. We can observe that all classification approaches are significantly better at identifying programming posts than at identifying

[5]https://swk-www.informatik.uni-hamburg.de/~soliman/ICSA2018.zip

TABLE III
EXPERIENCES OF PRACTITIONERS IN EXPERIMENT

Software development		Software architecture	
# Years	**# Participants**	**# Years**	**# Participants**
>12 Years	6	>5 Years	3
6-12 Years	6	2-5 Years	9
3-5 Years	4	1 Year	4

ARPs. Another observation is the ability of all classification approaches to identify "technology evaluation" ARPs better than other types of ARPs.

V. EVALUATION OF SEARCH APPROACH

The primary method to evaluate search approaches is expert judgment about the relevance of the results obtained from a search [11]. Therefore, to compare the effectiveness of the enhanced search approach proposed in this paper (ENHANCED) to a conventional keyword-based search (NORMAL), we conducted an experiment with 16 practitioners (Table III). Here, effectiveness is the ability of a search approach to identify posts relevant for solving architecture design tasks. Thus, we asked participants to perform architecture design tasks and use NORMAL and ENHANCED to search for information that supports these tasks. To implement NORMAL, we used the default implementation of Apache Lucene (i.e. no filtering/re-ranking of results). We tested the following hypotheses:

- H_0: There is no difference between the effectiveness of NORMAL and ENHANCED.
- H_1: There is a statistically significant difference between the effectiveness of NORMAL and ENHANCED.

A. Experiment Design

1) Experimental Corpus: We used a corpus of 2,511 posts from Soliman et al. [8] that were classified by practitioners. This corpus includes 1,653 programming posts, 282 technology identification posts, 291 technology evaluation posts and 285 features and configuration posts. Furthermore, we classified another 7,702 posts using the classification approach discussed in Section IV-E. These 7,702 posts were randomly selected from a larger pool of posts, which were obtained from Stack Overflow. Like with the 2,511 posts in the original corpus, we chose posts with positive user ratings on Stack Overflow to avoid poor quality posts in our corpus, because poor quality posts consume much time during a search [7]. This resulted in a total of 10,213 classified Stack Overflow posts as our experimental corpus. This number of posts is similar to other studies in software engineering that investigate search approaches for online communities (e.g. [28], [29]).

2) Architecture Design Tasks: Participants in the experiment solved six architecture design tasks, which required them to search for architectural information. Criteria for defining the design tasks were: a) Tasks are independent from the corpus of posts to ensure validity and to prevent bias, and b) tasks should simulate real design problems that occur during different design activities. Therefore, we interviewed three practitioners from different companies to provide real-world scenarios and related tasks, and for which they had to search for architecture information in the past. The interviewed practitioners were not participants in the experiment. This resulted in five tasks. We then categorized these five tasks according to the three design steps from Kazman and Cervantes [6] (see Section I-B). To include two tasks for each design step, we added a sixth task from a case study conducted by Kazman et al. [6]. Below we list a brief description for each task for each of the three design steps. A complete description for each task is available in our supplementary material.

Tasks for design step "identify design concepts":

- T1: For a realtime stock monitoring dashboard, identify middleware technologies which scale to > 100k users.
- T2: A claim management system needs to communicate with mobile apps. Identify JSON parsers for Java with high performance, and taking into consideration organizational policies around open source technologies.

Tasks for design step "select design concepts":

- T3: A help desk system communicates with a knowledge base via asynchronous communication and publish/subscribe patterns. Compare interoperability and latency of RabbitMQ, Apache Kafka, and ActiveMQ.
- T4: Compare three technology families for big data systems: Data collector, message brokers, and ETL engines; technologies must support a throughput of 15,000 events/second and ensure availability of 99.99%.

Tasks for design step "instantiate architecture elements":

- T5: CRM apps communicate with several other systems using Apache Camel and RabbitMQ. Search for technology features and components designs to determine mechanisms for message channeling, translation and routing, as well as a deployment topology (physical design).
- T6: An online shop in Java exposes services to other apps. Search for best practices regarding service decomposition to achieve high cohesion and low coupling.

To reduce bias or differences between the description of tasks, we captured tasks in a template. The template follows guidelines for developing simulated search tasks for information retrieval systems [30]. It includes a description of the functional and quality requirements, constraints, and the actual search goal of a task which aligns with the design steps.

3) Experiment Execution: We designed the experiment using a Graeco-Latin Square (GLS) design [31]. In this design, the order in which the six design tasks are performed as well as the order in which the two search approaches are used are rotated. Following the GLS design, there are six different sequences of tasks (T1, T2, T3,... T6, then T6, T1, T2,... T5, until sequence T2, T3,... T1). These six sequences are applied twice to rotate the order in which the two search approaches are used (i.e. in total there are 12 sequences of tasks). For the first six sequences (sequences T1, T2,... T6 to T2, T3,... T1), NORMAL was used for the first three tasks in a sequence and ENHANCED for the second three tasks in a sequence. Starting from the seventh sequence to the 12th sequence (i.e. again T1, T2,... T6 to T2, T3,... T1) ENHANCED was used for the first three tasks in a sequence and NORMAL for the second three tasks in a sequence. We randomly assigned participants

to sequences of tasks. Since we had 16 participants, every sequence was completed by at least one practitioner, and four sequences were completed by two practitioners.

Each participant read design tasks and then submitted search queries (sequences of words) and obtained a list of ranked posts. We asked participants to analyze only the top ten posts for relevance (users of search engines rarely look beyond the tenth result [32]) and rate each post on a Likert scale:

1) *Irrelevant (0)*: Post has nothing to do with the task.
2) *Low (1)*: Post contains information which is not immediately relevant to solving the task, but helps refine search.
3) *Medium (2)*: Post addresses a problem different but similar to task at hand, but still provides relevant information.
4) *High (3)*: Post addresses a similar or same problem as specified in the task and contains useful information to solve design task.

These ratings of participants for posts are needed to calculate measures for search engine effectiveness.

4) Measures of Effectiveness: We measure the effectiveness of each of the search engines using two metrics: $Precision@k$ and *Normalized Discount Cumulative Gain*($nDCG@k$), where k is the maximum number of posts that are considered for evaluation. We considered k from 1 to 10, because the top 10 search results are commonly checked by users of search engines [32].

$Precision@k$ [11] is the ratio between the number of relevant posts (low, medium or high), and the number of retrieved posts in results m, where $m \leq k$. m can be less than k if a poorly phrased query returns few results.

Precision does not consider the ranking of posts and their relevance. $nDCG@k$ [11] uses weights for posts based on their relevance and ranking in the list of search results (i.e. the higher the relevance and rank, the more weight). $nDCG@k$ for a query is calculated as $\frac{DCG}{IDCG}$, with $DCG = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)}$, where rel_i is the degree of relevance of a post found by a query (based on the Likert scale above). $IDCG$ is the DCG value for an ideal ranking of posts for a certain design task. In the above equation for $nDCG@k$, this ideal ranking is based on combining the individual rankings of posts (based on their relevance to a task) from all participants. For example, if user 1 rated posts x, y, z with relevance 3, 3, 1 for task T1 and user 2 rated posts a, b, c with relevance 2, 2, 1 for task T1, the ideal ranking for task T1 is 3, 3, 2, 2, 1, 1, 0, 0, 0, 0 when evaluating the top ten search results [11].

B. Evaluation Results

Participants submitted a total of 422 queries (on average 70 queries per task). The number of queries did not differ significantly between participants, tasks or design steps. The queries contain keywords, which refer to several AK concepts. For example, the query "Latency and reliability of RabbitMQ" contains keywords which refer to concepts like quality attributes and technology solutions. A complete list of executed queries is available in the supplementary material[5].

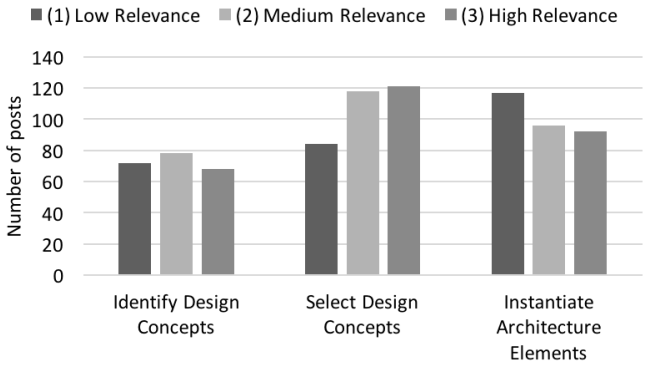

Fig. 3. Relevant posts identified by participants for each design step.

Fig. 3 shows the number of posts identified as relevant to tasks and grouped by design step. We note that queries related to tasks which belong to design step "select design concept" led to the highest number of posts and the highest relevance. Queries for tasks related to design step "identify design concepts" yielded the lowest number of posts.

We calculated $Precision@k$ and $nDCG@k$ for NORMAL and ENHANCED, with $@k_{1\rightarrow10}$. Fig. 4 shows a summary for all values of $nDCG$ as box-and-whisker plots $@k_{1\rightarrow3}$. The figure shows that ENHANCED has improved the overall nDCG values of searching. Fig. 5 shows the average $Precision@k$ and $nDCG@k$ for the three design activities and for ENHANCED and NORMAL. The ENHANCED approach improved the precision of the search for all three design activities. The "Select design concepts" design step achieves the highest $nDCG$ improvement, while step "Instantiate architecture elements" has the lowest improvement. Moreover, the values of $nDCG$ for "select design concepts" decrease gradually with the increase of k. A possible reason for this could be the ability of the classification approach to identify "Technology evaluation" ARPs better than "Features and configuration" ARPs (see Fig. 2).

To evaluate H_0 and ensure significance, we performed a two-sample T-test with unequal variances [33] on all values of $Precision@k$ of ENHANCED and NORMAL and $nDCG@k$ for ENHANCED and NORMAL, see Fig. 6. The results of the T-test indicate that ENHANCED outperforms NORMAL with statistical significance for both $Precision@k$ and $nDCG@k$ and for all values of k. From Fig. 6, we can observe that the T-values of $Precision$ start with high values (highest T-value for $Precision@k = 2$) and decrease gradually with the increase of k. On the other hand, the T-values of $nDCG$ start with a lower value $@k = 1$ and then increase $@k = 2$, and only change slightly until $@k = 10$.

VI. Discussion of Results and Future Work

A. Interpretation of Results

Normal search engines are limited for finding architecturally relevant information in communities. Even with a small search corpus and after removing low quality posts, the normal keyword-based search in our experiment could only reach an average $Precision@1$ (i.e. the first post in the returned list of

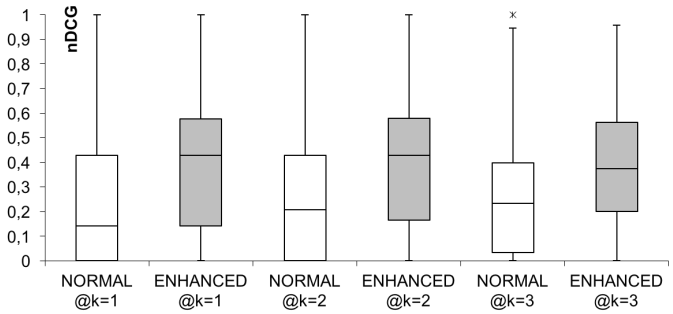

Fig. 4. $nDCG@1-3$ for NORMAL and ENHANCED. The lower and upper boxes are the lower and upper quartiles of the distribution, respectively. The vertical thin line is from the minimum to the maximum $nDCG$ values.

results is the most relevant post) between up to 0.51 and 0.62. A real search on millions of posts would yield lower precision. This lack of accurate search results from conventional searches explains the effort required to curate AK [5].

Our results show that filtering and re-ranking posts significantly improved the effectiveness of our search to find architecturally relevant information for design problems. In addition, the results show that searching for architectural information for different purposes (i.e. within different design steps) benefit differently from developer communities and the enhanced search. For example, our results in Fig. 3 and 5 show that tasks within the "Select design concept" design step find more relevant posts than other design steps. On the other hand, it is challenging to find posts for the "Identify design concepts" tasks. One reason behind this could be the abilities of the classification approach (see Fig. 2). Another possible reason is the complexity to describe a design problem using keywords, because the variations of terms for describing a design problem is big (e.g. domain and business terms).

The classification used in our search approach (Section IV-E) yields high accuracy when identifying and separating ARPs from programming posts (highest *F1*: 0.84 in Fig. 2). On the other hand, it is more challenging to further classify ARPs into their subtypes (highest *F1*: 0.55 in Fig 2 when classifying technology evaluation posts). The complexity to classify text on web pages according to certain architecture concepts has also been experienced by Gorton et al. [5], who achieved a maximum precision of 0.59 when detecting technology features in online technology documentation.

B. Threats to Validity

1) Internal validity: In our experiment with practitioners to evaluate the enhanced search, the experience and background of participants might have influenced the assessment of the architectural relevance of a post. Also, since participants completed the tasks in their own time, we did not have full control of their behavior (e.g. being tired). We tried to mitigate these issues through rotating the task order. Another threat is the design tasks. However, we used real design scenarios from practitioners and scenarios that were independent from the topics of the posts used for the experiment.

2) External validity: The effectiveness of the search approach (Section V-B) depends on the accuracy of the classification approach (Section IV-E). However, it is challenging to specify and generalize a clear relationship between the effectiveness of the search approach and the accuracy of the classification approach. The search approach depends on filtering and re-ranking ARPs as presented in Section III-A. When filtering ARPs, the classification approach produced good results (*F1*: 0.84 in Fig. 2). However, just filtering architecture posts is not sufficient to significantly improve the search for architectural information. In a separate pilot study (with different tasks and participants), we tested the search approach in Section III-A without re-ranking ARPs. The pilot study showed that filtering ARPs just slightly (not significantly) improved the effectiveness of search. Details and results of the pilot study are provided in the supplementary material[5]. However, the results of our experiments in Section V-B show that combining accurate filtering of ARPs with re-ranking ARPs has significantly improved the effectiveness of search, even with a low accuracy in classifying ARPs. Therefore, we expect even further improvements of the effectiveness of the search if the accuracy of the classification improves.

Re-ranking ARPs depends on "quota" assigned to each type of ARP, which is another threat. If these quotas change, then the search might identify other posts as architecturally relevant and the search might come up with different results. The lack of empirical studies on searching for architecture information using search engines prevented us from using mathematical modeling (e.g. [34]) for the re-ranking of posts. Mathematical modeling requires the execution and monitoring of thousands of queries using several tasks, which is challenging with the limited time available for architects to conduct experiments. To mitigate this threat, we followed an empirical approach and developed the quota based on an empirical study. Moreover, we fixed quotas for the experiment.

Regarding the evaluation of the search approach, the number of posts is limited due to the nature of our work. Parts of the corpus are classified by practitioners. This manual classification requires experienced practitioners rather than novices. It takes a practitioner around four hours to classify 100 posts. The same reason (time-consumption) prevents us from having a bigger number of participants for the experiment to evaluate the effectiveness of the search approach. It took participants between four and eight hours to solve the six tasks. This time is needed to understand the tasks, read posts carefully and get familiar with the domain. This is also why we had only two tasks per design step.

C. Future Work

During the experiments, practitioners faced problems related to the nature of architecture-relevant posts in developer communities. These problems deserve future work:

Complex semantics of discussions: Practitioners need to carefully read online discussions to understand described problems and their context. Even similar design problems often involve contextual differences. One reason for the textual complexity is incomplete information provided by users who cannot share the full context (e.g. business goals).

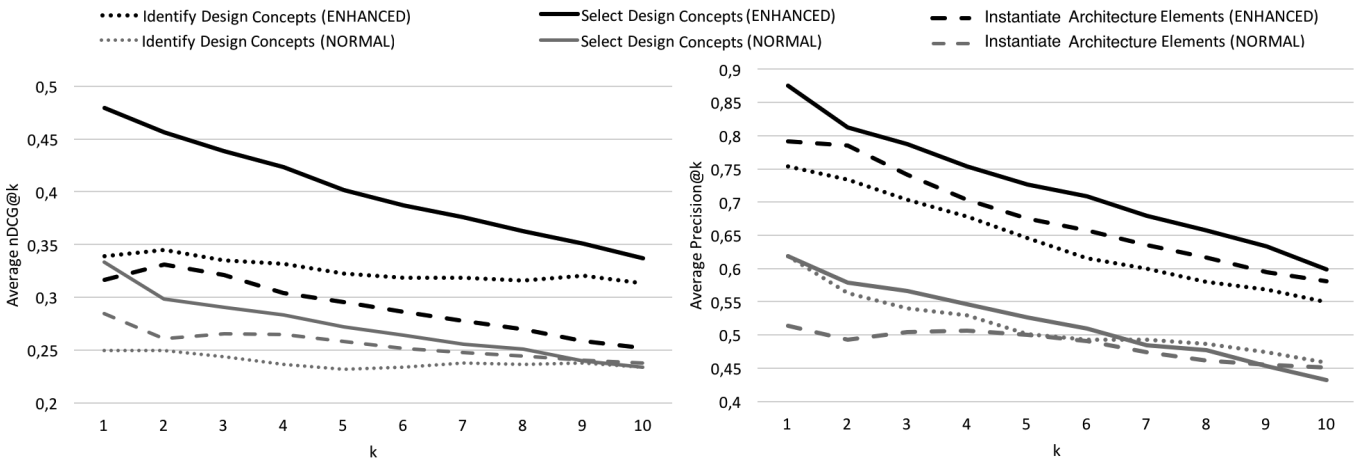

Fig. 5. Average $nDCG@k$ and $Precision@k$ for each design activity.

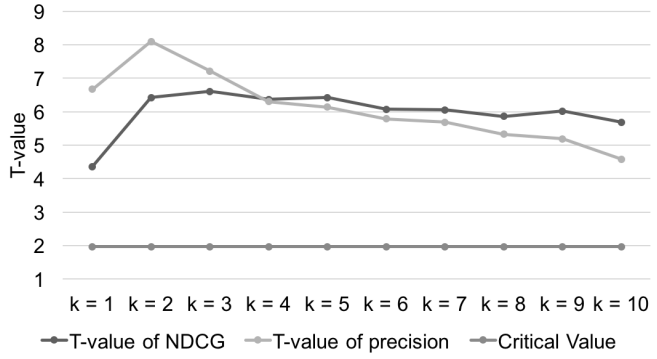

Fig. 6. Significance T-value for $nDCG@k$ and $Precision@k$. The critical T-value is 1.963 (T-values > 1.963 are considered significant). The higher the T-value, the higher the probability for being significant.

Required domain knowledge: When architects are not an expert in a problem domain, it becomes harder to specify useful and meaningful keywords for a search query. This is because developer community websites are not categorized according to the domain or types of design problems.

Lack of trust in community: Some of the information in developer communities could be wrong and misleading. Thus, architects need to be cautious when evaluating the proposed solutions and their descriptions and discussions.

VII. Related Work

The proposed approach is the first for a specialized search for architectural information in online developer communities. In this section, we give a brief overview on related work.

Capturing and documenting AK: During the last decade, approaches to capture AK have been proposed. A recent survey on AK [35] showed that most of the approaches to capture AK are about documenting design decisions in templates (e.g. [36]) or manually populating knowledge management systems (e.g. [37], [38]). Due to the effort required to document decisions manually, approaches have been proposed to support this process. For example, Falessi et al. [39] proposed to document only the most necessary decisions.

Automatically capturing AK: van der Ven et al. [40] proposed an approach to automatically analyze and capture information about design decisions from version management data of large open-source repositories. Lopez et al. [41] proposed an ontology-based natural language processing approach for capturing design decisions from existing architecture documents. Bhat et al. [42] proposed an approach to automatically identify and classify design decisions in issue management systems. However, existing approaches do not consider developer communities as a source for reusing AK.

Classification and search in developer communities: Within the last few years, developer communities have been studied. For example, Treude et al. [43] analysed programming posts on Stack Overflow qualitatively and defined several types (e.g. posts about debugging, how-to questions). Gottipati et al. [28] and Zou et al. [29] proposed approaches to improve the search for programming questions in software forums. The engines are based on classifying the discussions in online developer communities into semantically relevant categories using machine learning. However, these approaches did not investigate how online developer communities can be searched for architecturally relevant information.

AK in developer communities: Recent works explore AK in developer communities and products documentation web pages. In previous work [8], researchers analyzed Stack Overflow posts to determine types of architecture-relevant posts. In addition, previous work developed an ontology for the AK concepts, which are mentioned in Stack Overflow posts [9]. We used this ontology in our work. Gorton et al. [5] proposed an approach based on machine learning to automatically recommend web pages which contain AK relevant to certain technology features. The approach shows promising results for findings relevant architecture web pages. Our study complements these works towards specialized search approaches for software architecture in developers communities.

VIII. Conclusions

Effectively identifying architecture solutions to design problems became challenging due to the rapid change of architectural solutions (technologies, patterns, tactics, etc.). Recent

research efforts investigated online developer communities to deal with the challenges of architecture knowledge evolution. Our goal in this paper was to improve the search for AK in online developer communities based on architecturally relevant information in posts (rather than relying on keywords). We developed, implemented and evaluated an enhanced search approach for Stack Overflow. This search approach does not only rely on keywords, but also considers conceptual and semantic information about architecturally relevant information. Comparing this search approach with a conventional search approach showed that the enhanced search leads to more effective results when searching for information when identifying and selecting design concepts and instantiating architecture elements. Moreover, experiments with practitioners confirmed that searching and curating architecture knowledge from web pages using search engines is complex and time consuming. Most importantly, the results from experiments also showed improved effectiveness of searching based on the information needs for different architecture design steps.

REFERENCES

[1] 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*, vol. 19, no. 3, pp. 619–654, 2014.

[2] P. Kruchten, "Lifelong learning for lifelong employment," *IEEE Software*, vol. 32, no. 4, pp. 85–87, 2015.

[3] O. Zimmermann, J. Koehler, F. Leymann, R. Polley, and N. Schuster, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1249–1267, 2009.

[4] I. Gorton, J. Klein, and A. Nurgaliev, "Architecture knowledge for evaluating scalable databases," in *Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2015, pp. 95–104.

[5] I. Gorton, R. Xu, Y. Yang, H. Liu, and G. Zheng, "Experiments in curation: Towards machine-assisted construction of software architecture knowledge bases," in *IEEE/IFIP International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 79–88.

[6] R. Kazman and H. Cervantes, *Designing Software Architectures: A Practical Approach*. Addison-Wesley Professional, 2016.

[7] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" *Empirical Software Engineering*, vol. 22, no. 6, pp. 3149–3185, Dec 2017.

[8] M. Soliman, M. Galster, A. R. Salama, and M. Riebisch, "Architectural knowledge for technology decisions in developer communities: An exploratory study with stackoverflow," in *Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2016, pp. 128–133.

[9] M. Soliman, M. Galster, and M. Riebisch, "Developing an ontology for architecture knowledge from developer communities," in *International Conference on Software Architecture (ICSA)*. IEEE, 2017, pp. 89–92.

[10] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[11] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[12] M. Soliman, M. Riebisch, and U. Zdun, "Enriching architecture knowledge with technology design decisions," in *Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2015, pp. 135–144.

[13] L. Moreno, G. Bavota, S. Haiduc, M. Di Penta, R. Oliveto, B. Russo, and A. Marcus, "Query-based configuration of text retrieval solutions for software engineering tasks," in *10th Joint Meeting on Foundations of Software Engineering (FSE)*. ACM, 2015, pp. 567–578.

[14] B. Gleason, *The development of language*. Pearson Education, 2005.

[15] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[16] J. Fang, L. Guo, X. Wang, and N. Yang, "Ontology-based automatic classification and ranking for web documents," in *International Conference on Fuzzy Systems and Knowledge Discovery*. IEEE, 2007, pp. 627–631.

[17] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1, pp. 1–39, 2010.

[18] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[19] Z. Xing, J. Pei, and E. Keogh, "A brief survey on sequence classification," *SIGKDD Explorations*, vol. 12, no. 1, pp. 40–48, 2010.

[20] J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas, "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226–239, 1998.

[21] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[22] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[23] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[24] C. Treude and M. P. Robillard, "Augmenting api documentation with insights from stack overflow," in *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. ACM, 2016, pp. 392–403.

[25] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, no. 2, pp. 131–163, 1997.

[26] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Machine Learning*, vol. 59, no. 1-2, pp. 161–205, 2005.

[27] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Eleventh Conference on Uncertainty in Artificial Intelligence (UAI)*. Morgan-Kaufmann, 1995, pp. 338–345.

[28] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2011, pp. 323–332.

[29] Y. Zou, T. Ye, Y. Lu, J. Mylopoulos, and L. Zhang, "Learning to rank for question-oriented software text retrieval (t)," in *International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 1–11.

[30] P. Borlund, "The iir evaluation model: a framework for evaluation of interactive information retrieval systems," *Information Research*, vol. 8, no. 3, 2003.

[31] J. T. Sutcliffe, "The pragmatics of information retrieval experimentation, revisited," *Information Processing and Management*, vol. 28, no. 4, pp. 467 – 490, 1992.

[32] L. A. Granka, T. Joachims, and G. Gay, "Eye-tracking analysis of user behavior in www search," in *International Conference on Research and Development in Information Retrieval*. ACM, 2004, pp. 478–479.

[33] D. C. Montgomery, *Design and Analysis of Experiments*. John Wiley & Sons, 2006.

[34] M. Fernández, D. Vallet, and P. Castells, "Probabilistic score normalization for rank aggregation," in *Advances in Information Retrieval*. Springer, 2006, pp. 553–556.

[35] D. Tofan, M. Galster, P. Avgeriou, and W. Schuitema, "Past and future of software architectural decisions - a systematic mapping study," *Information and Software Technology*, vol. 56, pp. 850–872, 2014.

[36] J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, vol. 22, no. 2, pp. 19–27, 2005.

[37] M. Babar, I. Gorton, and B. Kitchenham, "A framework for supporting architecture knowledge and rationale management," in *Rationale Management in Software Engineering*. Springer, 2006, pp. 237–254.

[38] U. van Heesch, P. Avgeriou, and R. Hilliard, "A documentation framework for architecture decisions." *Journal of Systems and Software*, vol. 85, no. 4, pp. 795–820, 2012.

[39] D. Falessi, L. C. Briand, G. Cantone, R. Capilla, and P. Kruchten, "The value of design rationale information," *ACM Transactions on Software Engineering Methodology*, vol. 22, no. 3, pp. 21:1–21:32, 2013.

[40] J. S. van der Ven and J. Bosch, "Making the right decision: Supporting architects with design decision data," in *7th European conference on Software Architecture (ECSA)*. Springer, 2013, pp. 176–183.

[41] C. Lopez, V. Codocedo, H. Astudillo, and L. M. Cysneiros, "Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach," *Science of Computer Programming*, vol. 77, no. 1, pp. 66 – 80, 2012.

[42] M. Bhat, K. Shumaiev, A. Biesdorf, U. Hohenstein, and F. Matthes, "Automatic extraction of design decisions from issue management systems: A machine learning based approach," in *Software Architecture*, A. Lopes and R. de Lemos, Eds. Springer, 2017, pp. 138–154.

[43] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (nier track)," in *International Conference on Software Engineering (ICSE)*. ACM, 2011, pp. 804–807.