

Probing for Requirements Knowledge to Stimulate Architectural Thinking

Preethu Rose Anish¹, Balaji Balasubramaniam¹, Abhishek Sainani¹,
Jane Cleland-Huang², Maya Daneva³, Roel J. Wieringa³, Smita Ghaisas¹
TATA Research Development and Design Centre (TRDDC)¹, TATA Consultancy Services Ltd., Pune, India
{preethu.rose, balaji.balasubramaniam, a.sainani, smita.ghaisas}@tcs.com
Systems and Requirements Engineering Center (SAREC)² DePaul University, Chicago, USA
{jhuang}@cs.depaul.edu
University of Twente³, Enschede, the Netherlands
{m.daneva, r.j.wieringa}@utwente.nl

ABSTRACT

Software requirements specifications (SRSs) often lack the detail needed to make informed architectural decisions. Architects therefore either make assumptions, which can lead to incorrect decisions, or conduct additional stakeholder interviews, resulting in potential project delays. We previously observed that software architects ask Probing Questions (PQs) to gather information crucial to architectural decision-making. Our goal is to equip Business Analysts with appropriate PQs so that they can ask these questions themselves. We report a new study with over 40 experienced architects to identify reusable PQs for five areas of functionality and organize them into structured flows. These PQ-flows can be used by Business Analysts to elicit and specify architecturally relevant information. Additionally, we leverage machine learning techniques to determine when a PQ-flow is appropriate for use in a project, and to annotate individual PQs with relevant information extracted from the existing SRS. We trained and evaluated our approach on over 8,000 individual requirements from 114 requirements specifications and also conducted a pilot study to validate its usefulness.

CCS Concepts

• General and Reference→Surveys and overviews, Empirical studies, Evaluation, Experimentation, Validation •Software and its engineering→Software architecture, Requirements engineering •Information systems→Information retrieval, Machine learning

Keywords

Architecturally significant requirements, automated requirement classification, functional requirements, requirements knowledge, Probing Questions (PQs), PQ-flows.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

ICSE '16, May 14-22, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-3900-1/16/05...\$15.00
DOI: <http://dx.doi.org/10.1145/2884781.2884801>.

1. INTRODUCTION

Software requirements specifications (SRS) often lack the detail needed by Software Architects (SAs) to make informed architectural decisions [2, 3]. For example, a SRS might include a high level ‘Auditing’ requirement, but fail to specify whether the audit is for internal or regulatory compliance, what data must be produced for compliance, or if notification messages must be sent to stakeholders when certain audit events occur. Such details impact architectural decisions and should be exposed early in the requirements elicitation and analysis process. If wrong architectural decisions are made, the intended but unstated functional requirements will not be satisfied. In practice, customers often assume that developers implicitly understand their needs and therefore do not fully express their requirements [1]. Experienced SAs intuitively realize when critical information is missing. To compensate, they may make assumptions based on their own knowledge of the problem or conduct additional formal or informal interviews with the stakeholders to seek clarification and to elicit the missing information. However, assumptions, if incorrect, can lead to costly and time-consuming refactoring efforts at later stages of projects; while additional unplanned interviews can slow down time-to-market. Asking business analysts (BAs) to provide richer, more informative specifications may seem like a good idea, but it is unlikely to be effective, given that BAs typically lack technical architectural knowledge. They are often seen to be ill-equipped to ask the right kind of questions needed to unearth architecturally significant information.

In our previous study, we found that experienced SAs address missing information by asking Probing Questions (PQs) [2, 3]. In this paper, we systematically elicit, structure, and document these PQs in order to make them available for use by BAs. In the Requirements Engineering and Software Architecture literature, it is well understood that both Functional and Non-Functional Requirements (NFRs¹) can have an impact on architectural design. However, we focus our efforts in this paper on functional requirements, because (1) the architectural impact of functional requirements is often implicit unlike that of NFRs, which explicitly solicit architectural considerations and (2) much of the published literature [for example: 4 - 9] on architecturally significant requirements has focused on NFRs. Far less emphasis

¹ As there is no commonly agreed umbrella term for such requirements, we have opted to use the traditional term of NFR, which is broadly understood in both industry and academia


has been placed on addressing the architectural significance of functional requirements [10].

Throughout the remainder of the paper, we refer to functional requirements with architectural significance as ‘Architecturally Significant Functional Requirements (ASFRs)’. While almost all functional requirements (FRs) may be argued to have some degree of impact on software architecture; our focus is on those FRs that have a *significant* impact. By this, we mean any FR that (i) has a potentially broad impact across the system, (ii) is high-risk, possibly volatile, and (iii) if modified could involve expensive refactoring [2, 3].

The primary goal of our work is to leverage the knowledge of experienced SAs and to make it available to BAs so that they are equipped to elicit a more complete set of requirements that feed sufficient information into the architecture design process. While Software Engineering scholars agree that requirements elicitation and architectural design should be performed iteratively [10, 11, 12], little is known about what specifically BAs could or should do, in order to jump-start the process by identifying customers’ requirements in a way that helps SAs more effectively evaluate candidate architectural solutions.

To this end, we collected and analyzed the perceptions of more than 40 experienced SAs on the process of identifying, analyzing, organizing, and evaluating PQs for five commonly occurring functional areas namely: *Audit Trail*, *Batch Processing*, *Business Process State Alerts*, *Report* and *Workflow*. We previously defined these [3] as follows:

Audit Trail: Audit Trail facilitates auditing of system execution. Depending on the purpose of the audit, architecture can change. For example, an audit for a regulatory compliance purpose would imply provisioning for data lineage.

 **Processing:** This includes requirements facilitating batch processing. Most batch processes entail assigning exclusive access privileges to stakeholders in charge of executing them.

Business Process “State” Alerts: This class of ASFR is concerned with notifications, generated often as part of executing a business process by means of the respective system. If a response to such an alert is expected, it would entail a different design choice than a simple unilateral alert message.

Report: This category includes FRs that facilitate Report generation. The architectural decisions for a report intended to feed into business analytics involve provision to process structured and/or unstructured data.

Workflow: This includes requirements that provide support to move work items, facilitate reviews and approvals. Complexities associated with workflows need to be understood upfront in order to avoid expensive rework at later stages of projects.

These areas were selected because (a) they occur commonly across many different systems, and (b) they emerged as important topics of architectural significance in an earlier study we had conducted [3]. The resulting PQ-flows are the first contribution of this paper and can be used as catalogs by BAs in requirements elicitation. We note that the survey participants were from various multi-national companies in which BA and SA are separate disciplines and where the two sets of skills - requirements knowledge and architectural knowledge, typically reside with different people.

Our second contribution is our automated approach for analyzing a software requirements specification (SRS) to determine if a PQ-

flow is relevant, and then annotating the PQs with relevant information found in the SRS. The annotated PQ-flow then provides an appropriate structuring of questions to ask, while depicting relevant answers (and other supporting information) that are already available in the SRS. In addition to presenting results from our data collection and survey with the SAs, we report a series of experiments which were conducted to evaluate the effectiveness of annotating the PQ-flows. We show that PQ-flows can lead to improved and more complete SRSs.

The remainder of the paper is organized as follows: Section 2 presents background information, Sections 3 and 4 report on the construction of the PQ-flows and describe our approach for contextualizing them within a project. Section 5 then presents an initial user evaluation of our approach. Finally, Sections 6 to 8 present threats to validity, related work, and conclusions.

2. BACKGROUND

This work is motivated by the findings from our previous qualitative interview-based study [2, 3] which we briefly summarize here. The study [3] explored current practices for working with ASFRs as perceived by 14 SAs from large organizations in India, the United States and the Netherlands. The participating SAs were from the domains of Banking, Financial Services and Insurance (BFSI), Healthcare, Retail, Government, Telecommunications, and Airlines Management. Our key findings were that (1) SRSs often lack crucial architecturally relevant information needed to make informed architectural decisions, (2) PQs are an essential mechanism for unearthing underspecified architecturally relevant information. Our study identified a total of 15 categories of ASFRs, each with its own set of PQs. From these we selected the five ASFR categories which are the focus of this paper.

Our study also suggested that there was a logical way to sequence them during the elicitation process. We observed that experienced SAs understand what PQs need to be asked, have a tacit conceptual mental model of the interdependencies between the PQs, and therefore sequence their questions in a meaningful way. We note that similar observations were made by Li et al., who reported that much architectural knowledge is tacit in nature and should be recovered and documented [13]. If SAs’ tacit knowledge on the order of use of the PQs becomes explicit and shared with BAs, creating a repository of PQ-flows for each ASFR category could generate a body of knowledge similar to the Design Patterns of Gamma et al. [14] and be used by BAs to enhance the quality and completeness of SRSs. While the focus of our work is on ASFRs from the application domains mentioned earlier, we expect that PQ-flows could similarly be documented and made available for ASFRs from other areas.

3. CONSTRUCTION OF PQ-FLOWS

To discover and document PQ-flow structures for the five selected ASFR categories, we followed a research process that was inspired by the design science approach [15]. We started with a ‘basic’ design of the PQ-flow and then integrated practitioners’ feedback into the design. This resulted in a refined version of the PQ-flows for which further feedback was elicited. The process included the following steps: (a) interview SAs, (b) analyze the PQs obtained through the interviews and create the initial basic PQ-flows in consultation with a SA, (c) conduct an online survey (henceforth referred to as Phase 1 of the survey) to collect feedback on the early PQ-flows that were created in step (b), refine the flows, and (d) conduct a follow up online survey (henceforth referred to as Phase 2 of the survey) to evaluate the

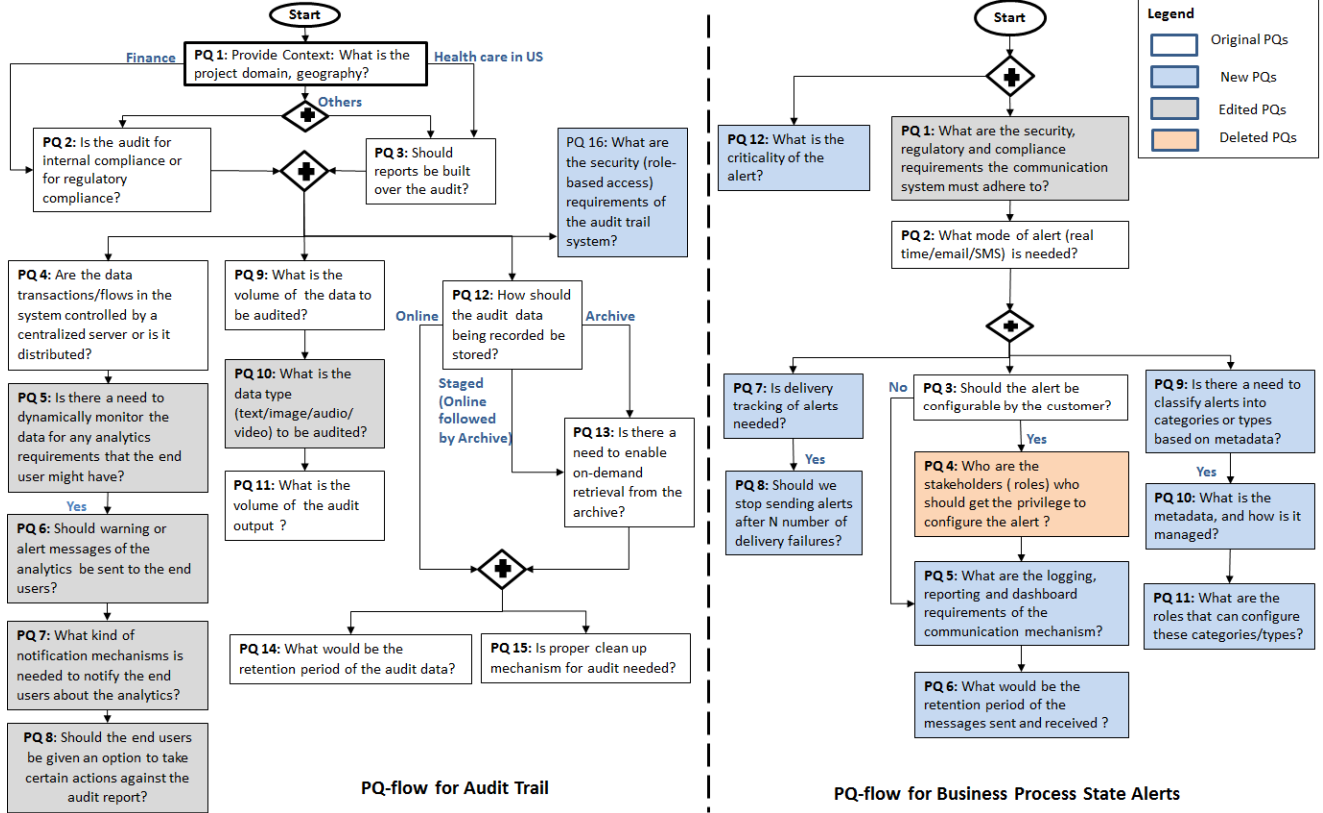


Figure 1: PQ-flows for Audit Trail (left) and Business Process State Alerts (right)

refined PQ-flows from step (c). Steps (a) and (b) were helpful in designing the basic PQ-flows for the five ASFR categories, while steps (c) and (d) were feedback-collection exercises which helped to refine the basic designs and to remove any bias that might have been passed into them. We now explain each of these steps in more detail.

3.1 Interview with Software Architects

Our prior interviews with SAs [2, 3] provided a basic unstructured list of PQs for each ASFR category. Two of the authors of this paper therefore manually analyzed the PQs to identify a candidate logical sequence of questions. For instance, in case of *Audit Trail* category, we found that the PQ “What kind of notification mechanisms are needed to notify the end user about the analytics?” should be asked only if the customer first answered “Yes” to the PQ “Should warning or alert messages of the analytics be sent to the end users?”. Similarly, if the domain under consideration is Finance, it is not relevant to ask the PQ “Should reports be built over the audit?”. This is because as per our interview participants “if the domain is finance, it is obvious that reports are needed”.

Once the basic PQ-flow structures for the five ASFR categories were identified, we carried out a series of consultations with a senior SA who had more than 20 years of experience in Banking, Healthcare and Insurance domains. This was done to collect the SA’s feedback on the proposed PQ-flows. The consultation included a joint review of the basic PQ-flows for each ASFR category and a walk-through session in which the SA played the role of both customer and SA and provided insights into the contextual settings under which he might ask a particular PQ as

well as the effect of the customer’s answers on the PQ-flow sequence. This exploratory exercise yielded five refined PQ-flows that formed our initial PQ-flow catalog.

3.2 The Online Survey

We used the online survey method [16] to validate and improve the initial PQ-flow structure in step (b). This was essential to reduce possible bias of the authors and the senior SA. The use of a survey method enabled the inclusion of large number of participants which in turn enabled us to recognize trends in responses as the design of the PQ-flows was incrementally improved. The objective of our survey was to evaluate the initial PQ-flows, identify missing questions to improve the completeness of the PQ-flows, modify or remove invalid PQs, and improve their logical order.

The number of questions in the online survey was kept as small as possible in an effort to encourage participants to complete the entire survey. The survey was implemented using Survey Monkey. Interested readers can refer to the questionnaire on our project page <http://re.cs.depaul.edu/ASFR>

We piloted the questionnaire with two SAs and integrated their feedback. The feedback was used primarily to improve user-interactions (i.e. through improved instructions). As the survey questions were only slightly modified, we included the responses from the pilot study in the final analysis.

The survey targeted professionals who have worked in the capacity of SAs in organizations ranging from multi-national companies to start-ups. As an incentive for participating in the study, we offered to share the final report of the results from our

work on PQs. The questionnaire captured demographic information concerning the SA's experience and primary business domain. As depicted in Table 1, over 73% of our participants had more than ten years of IT experience, while over 40% had at least five years' experience as a SA. The primary business domains of the respondents included Energy, Resources and Utilities (ERUG), Media and Information Services, Banking, Financial Services and Insurance (BFSI), Education, Content Management, Healthcare, Retail, Automotive, Telecommunication, Analytics, Recruitment and ERP for Manufacturing, Real estate, Hospitals.

For each of the five ASFR categories, the survey presented a diagram, similar to those shown in Figure 1, depicting the PQ-flows. The participant was then asked to evaluate the PQ-flow and to identify missing PQs (if any) and those that should be modified, deleted or re-sequenced. The participants' responses were then used to refine the PQ-flow for each ASFR category.

Table 1. Demographic details of the respondents

TOTAL YEARS OF IT EXPERIENCE		TOTAL YEARS OF EXPERIENCE AS A SA	
Choices	Response	Choices	Response
< 1 year	0.00 %	< 1 year	10.53 %
1 to 5 years	10.53 %	1 to 5 years	47.37 %
5 to 10 years	15.79 %	5 to 10 years	36.84 %
> 10 years	73.68 %	> 10 years	5.26%

3.2.1 Online Survey Phase 1 and Analysis

Phase 1 of the online survey was carried out over a period of 3 weeks (03rd July – 24th July, 2015). We received a total of 22 responses. The survey was qualitative in nature and therefore once the survey responses were received, we performed qualitative data analysis [17] following the guidelines of Charmaz' Constructive Grounded Theory method [18]. In this method, different concepts and categories are extracted from the instances in the textual data. The data analysis was performed by three of the authors (PRA, AS and SG)². Based on the findings from our data analysis, we again refined the PQ-flows for each of the five ASFR categories. This was done as follows: three of the authors of this paper analyzed all the responses and then for each ASFR category, organized the PQs in sequence and logically appended them to the existing PQ-flows. While merging the feedback from different SAs, there were some minor cases of conflict such as cases where a missing PQ was specified by different SAs in different ways and/or at different levels of granularity. In such cases the authors discussed amongst themselves and chose the most appropriate PQ form to append. Also, there were several instances of agreements. For example, for the *Workflow* category, two of the Participants (P3 and P7) suggested we exclude the PQ: *Do you want to have support for sub-workflow?* They gave the following explanation:

P3: "How does a user know that they need a sub workflow? It's more of a technical implementation. This PQ can be removed."

P7: "A decade back, it might be an issue, but with many frameworks offering this as out-of-shelf, I don't think we should really focus on this."

We rejected some of the feedback comments for the reasons explained below:

- Some suggestions for adding new PQs were not accepted (i) because they pertained to design considerations rather than

architecturally significant PQs. For example, for *Report* category, Participant P 12 suggested that we add the PQ: "What are the filters needed for the report?" or (ii) because those PQs were already covered in the existing PQ-flows at different levels of granularity. We attempted as much as possible to maintain similar level of granularity across all PQs.

- Some suggestions for deleting PQs were not accepted either. For instance, Participant P16 suggested deleting several PQs from *Audit Trail*: "Delete PQ 12 through PQ 15 as these don't strictly fall in audit trail category. Audit trails are typically "passive". PQ 12 through 15 is venturing into monitoring and further into actionizing." PQ 12 through PQ 15 for *Audit Trail* talked about dynamically monitoring the data transactions/flows to detect abnormal patterns and notification to be sent for such patterns. However, Participant P4 mentioned the following for the same PQs: "I think PQ 12 through PQ 15 should be modified to include any analytics requirements that the customer might have instead of only looking for abnormal patterns. The audited data can provide a mine of information". Analyzing these two feedback comments, we included the suggestion by P4. Instead of deleting PQ12 through PQ15, we modified them as per Participant P4's suggestion.

Conflicts among the three authors (if any) were resolved by discussion amongst the authors, and, when necessary contacting the survey respondents for clarifications.

The modified PQ-flows were then evaluated in Phase 2 of the online survey, using the same questionnaire as the one in Phase 1.

3.2.2 Online Survey Phase 2 and Analysis – Enriching the PQ-flows

Phase 2 of the online survey was conducted over a period of 2 weeks (31st July – 14th August, 2015). The responses were analyzed using the same qualitative analysis techniques used in Phase 1. We received 10 responses in Phase 2. We note that the respondents in Phase 2 were different from the respondents in Phase 1 of the survey. This time, the analysis of responses indicated very few requests for modifications, additions, deletions, or re-organization of the PQ-flows. This suggests that the existing catalog of PQ-flows adequately covered most of the important architectural concerns from these architect's viewpoint.

Figure 1 depicts the PQ-flows for *Audit Trail* and *Business Process "State" Alert*. Table 2 depicts the count for the number of PQs added to, edited and deleted from the original PQ list (since the beginning of Phase 1 of the survey) for each of the five ASFR categories at the completion of Phase 2 of the survey. An online catalog of all the five PQ-flows can be viewed at our project page <http://re.cs.depaul.edu/ASFR>

Table 2: Towards enriched PQ-flows

Category	No. of PQs added	No. of PQs edited	No. of PQs deleted
Audit Trail	7	4	1
Batch Processing	4	5	0
BPSA	4	4	0
Report	12	1	1
Workflow	8	0	0

4. CONTEXTUALIZING PQ-FLOWS

Now that the catalog of PQ-flows is defined, we focus on the application of a specific PQ-flow within a project. This is to move

² PRA – P.R. Anish, AS – A. Sainani, SG – S. Ghaisas

to the second contribution of this paper, namely automated support for determining relevance of a PQ-flow for a SRS, and automated annotation of relevant PQs with information found in the SRS. Our first goal is to determine when a flow is relevant to the project i.e. determining if there is any evidence in the existing requirements that *Audit Trail*, *Workflow*, or any other ASFR category is relevant. Our second goal is to identify requirements that already contain details sought by questions in the relevant PQ-flow and then to annotate the PQ-flow diagram accordingly. The BAs can leverage this information to determine what is already known, what knowledge is missing, and what effect this would have on the flow of the PQs. For example, if the following requirement is included in the SRS: “*For regulatory compliance, an audit trail of all the transactions should be maintained*”, then there is a clear indication that the audit is for regulatory compliance. In this case, there would be no need to ask the PQ: “*Is the audit for internal compliance or for regulatory compliance?*”

Figure 2 depicts our overall approach for contextualizing the PQ-flows. It shows the initial step of scanning the SRS in order to identify ASFRs, recommending the use of specific PQ-flows, and then annotating the PQ-flow diagram with relevant requirements which already contain details sought by the specific questions in the PQ-flow. In the following sub-sections we elaborate on each of these activities.

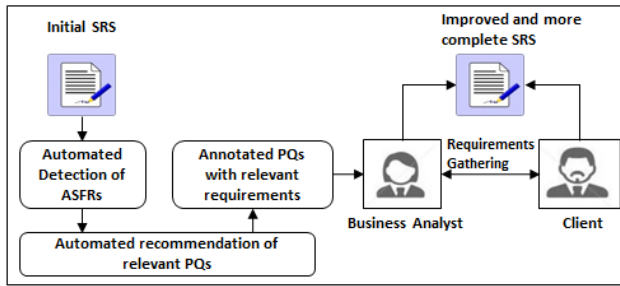


Figure 2: Contextualizing the PQ-flows

4.1 Detecting ASFRs

The first challenge is to determine which of the PQ-flows are relevant for a given project context. To this end we have trained a classifier to recognize requirements related to each of the five ASFR categories that we report about in this paper. If such requirements are found in a SRS, then we assume the corresponding PQ-flow is relevant.

In our earlier work [2], we reported results from a preliminary proof-of-concept experiment in which we trained a classifier to detect 10 different categories of ASFRs. However, the classifier was trained using only 450 functional requirements including a total of 246 ASFRs dispersed across 10 categories (i.e. *Audit Trail*, *Batch Processing*, *Business Process “State” Alert (BPSA)*, *Localization*, *Payment*, *Print*, *Report*, *Search*, *Third Party Interaction (TPI)* and *Workflow*). Furthermore, it suffered from the major limitation that each requirement was classified into exactly one class, whereas approximately 20% of requirements in our studied SRS were manually classified as belonging to two or more classes. For example, the requirement stating that the “*system should generate reports on the complaint register on a monthly basis through a batch process*” clearly provides information about both *Reporting* and *Batch Processing*. The work reported in this paper addresses these issues through retraining the classifier on a far larger number of requirements and utilizing a classifier which supports multi-classification.

4.1.1 Training on larger set of ASFRs

To increase reliability of our approach and to scale it up for use in large industrial projects, we retrained the classifier against 8,669 requirements statements dispersed across 114 distinct SRSs from the insurance domain. While the remainder of this paper focuses on only five ASFRs, we trained the classifier for a broader set of requirements classes and report the results here.

Three of the researchers (PRA, BB, AS)³ manually analyzed the 8,669 requirements to label each statement as an ASFR or a non-ASFR and then classified each one into one or more of the ASFR categories. Each researcher individually reviewed and classified a third of the requirements. If there was any doubt about the class of a requirement, all three researchers discussed together to arrive at a consensus. Furthermore, the researchers periodically reviewed each other’s work to ensure ongoing agreement. The resulting dataset formed the ground truth against which the identification and classification results could be compared. The task took a total of 60 hours to complete.

Out of the 8,669 requirements, the counts attributed to each ASFR category ranged from 2 for *Online Help* to 424 for *Localization/Multilingual*. Since *Online Help* had negligible instances (i.e. only 2), we did not include it for the ASFR categorization step.

4.1.2 ASFR Identification

Our classification approach includes two steps. In this first step, we simply classify each requirement as ASFR or non-ASFR. In a preliminary informal analysis, we compared the efficacy of using Naïve Bayes, J48 decision tree, Support Vector Machine, Nearest Neighbor and AdaBoost and found that Naïve Bayes performed the best on our data. This matches results observed in other studies related to classification problems [51]. We therefore adopted Naïve Bayes [19] for the classification task. The Naïve Bayes classifier is a probabilistic classifier based on Bayes theorem and follows the assumption that the contribution of an individual feature towards deciding the probability of a particular class is independent of other features in that dataset instance. For example, in the case of detecting an ASFR class, the contribution of the term ‘log’ is independent of the contribution of the term ‘notification’. Naive Bayes tends to overestimate the probability of the selected class. However, since we use it only to predict the presence of the class and not its probability, it produces quite accurate results.

Because instances of ASFRs and non-ASFRs were imbalanced (i.e. 1434 ASFRs and 7235 non-ASFRs), we adopted stratified random sub-sampling. The majority class (i.e. the non-ASFRs) was under-sampled in each iteration. A disadvantage associated with under-sampling is that it is known to result in loss of information [20]. To counter this, we repeated the experiment such that all non-ASFRs were validated at least once. We used 5-fold cross validation to divide the data into five evenly sized buckets, such that instances of each ASFR and non-ASFR were distributed evenly across the buckets. In each iteration, one bucket was set aside for testing and the remaining four buckets were used to create a training set. The sub-sampling was applied to equalize the instances of ASFRs and non-ASFRs and then used to train the Naïve Bayes classifier. The trained classifier was then used to classify the requirements in the test set. This process was repeated 5 times, until all data had been classified.

³ PRA – P.R. Anish, BB – B. Balasubramaniam, AS – A. Sainani

All requirements were pre-processed in WEKA to remove stop words, and to stem each word to its morphological root using Lovins' stemming algorithm [21]. The resulting terms formed the feature attributes. In order to select feature attributes that would help in classification, we used the technique of Information Gain [22] in WEKA to rank these features according to their relevance to classification. We then selected the top 100 feature attributes for our classification experiments.

Our approach achieved an average recall of 0.81 and an average precision of 0.69.

4.1.3 Categorizing ASFRs by Type

The second task involved classifying the identified ASFRs into specific categories. Classification at this level of granularity is necessary for determining which PQ-flows are relevant to a given project, and as a precursor for attaching PQs to a given ASFR.

Because an ASFR can belong to more than one class, we investigated classifiers which are capable of performing multi-label classification. In particular, we performed initial exploratory investigations with the RAndom k labELsets classifier (RAkEL) [23], Naïve Bayes Classifier [19] and Nearest Neighbor Classifier [24] using the **Mulan software package** [25]. Mulan has specialized algorithms to handle multi-label classification problems. In this paper, we report our results with RAkEL since it outperformed the other classifiers in the case of our dataset and the optimization parameters chosen. Moreover, during the manual analysis of our dataset, we discovered some label correlations in our dataset. For instance, labels 'workflow' and 'third party interaction' occur together for certain statements. RAkEL also incorporates such correlations during the classification task. RAkEL is an enhancement of the Label Powerset (LP) method that incorporates label correlations by considering each subset of labels (called labelsets) that exists in the training set, as a different class value in a single-label classification task. However, if the training set has a large number of labelsets, LP suffers from higher computational cost and the dataset can be skewed if some labelsets have limited training examples. Moreover, during the testing phase, LP can predict only the labelsets learned from the training set. RAkEL overcomes the limitations of LP by randomly breaking the initial set of labels into a number of small-sized labelsets and then training a multi-label classifier on the training set with those labelsets using LP. The k in RAkEL is a parameter for the size of the labelsets.

We present the results for each of the evaluated ASFR categories in Table 3, using four metrics: Precision, Recall, F-Measure, and Specificity. Recall measures the fraction of actual ASFR instances which are correctly assigned, precision measures the fraction of classified instances which are correct, F-Score is the harmonic mean of recall and precision, and specificity measures the extent to which the algorithm rejected instances that were not of the specified category. The formal definitions of these metrics are provided in [26, 27].

4.1.4 Analysis of results

The results reported in Table 3 show that the greatest accuracy was observed for *Batch Processing*, *Print*, *Localization/Multilingual* and *Search*, which all achieved F-Scores above 0.90. Overall, F-score values ranged from 0.17 to 0.94. A consistently good specificity of 0.95 and above for all the categories indicates that the classifier successfully avoided many misclassifications.

The lower accuracy for some categories such as *Storage Mechanism* and *Payment* can be attributed to the fact that they had very few instances (26 and 41 respectively). In fact, our prior

Table 3. Classification results for each ASFR category at statement level

ASFR Category	#Statements	Precision	Recall	F-score	Specificity
Audit Trail	89	0.81	0.85	0.82	0.98
Batch Process	89	0.90	0.96	0.93	0.99
BPSA	246	0.76	0.60	0.67	0.96
Localization/Multilingual	424	0.93	0.95	0.94	0.97
Payment	26	0.50	0.20	0.26	0.99
Print	164	0.90	0.96	0.93	0.98
Report	90	0.97	0.83	0.89	0.99
Search	142	0.95	0.88	0.91	0.99
Storage Mechanism	41	0.36	0.12	0.17	0.99
TPI	281	0.76	0.62	0.68	0.95
Workflow	81	0.78	0.52	0.58	0.99
AVERAGE		0.79	0.69	0.71	0.98

work has shown that increasing the size of the training set can improve the accuracy of requirements classification results [28]. For some categories such as *BPSA* (*Business Process "State" Alert*) and *TPI* (*Third Party Interaction*), accuracy was also relatively low despite the fact that the categories were well represented at 246 and 281 respectively. We observed that requirements in these two categories tended to belong to a higher-than average number of class associations. For instance, the requirement statement "*There should be a monthly batch process that generates a report of premium defaulters and then triggers an email reminder for the defaulter policy holders. This requirement is only for Denmark*" falls into 4 ASFR categories namely *Batch Process*, *BPSI*, *Localization/Multilingual* and *Report*. Our analysis of the results suggested that the results from the classifier were less accurate for such requirements.

4.2 Recommending Relevant PQ-flows

A specific PQ-flow is recommended to a BA for use in a project if, and only if, the current SRS contains one or more ASFRs of the respective category. For example, the *Audit Trail* PQ-flow would be recommended if at least one audit-related requirement is found in the SRS. We therefore analyzed the classification results at the SRS level. We refer to this approach as Relevance algorithm.

Based on the previously established ground truth (see Section 4.1.1) and results from section 4.1.3, we determined whether a given PQ-flow was relevant to each one of the 114 SRS documents. We evaluated the relevance algorithm using Precision, Recall, Specificity, and F-score results accordingly.

Table 4 depicts results from the relevance algorithm for each of the five ASFR categories in the 114 SRS documents. The results from our analysis show that we were able to quite accurately determine when to recommend a PQ-flow for use in a project. Precision ranged from 0.92 (*Workflow*) to 0.98 (*BPSA*) while recall ranged from 0.79 (*Workflow*) to 0.10 (*Batch Processing*). This indicates that the approach is capable of retrieving most of the relevant SRSs for the PQ-flows for the ASFR categories. Specificity ranged from 0.93 (*BPSA*) to 0.98 (*Workflow*), indicating that there were few cases in which we failed to recommend relevant PQ-flows, or made incorrect recommendations.

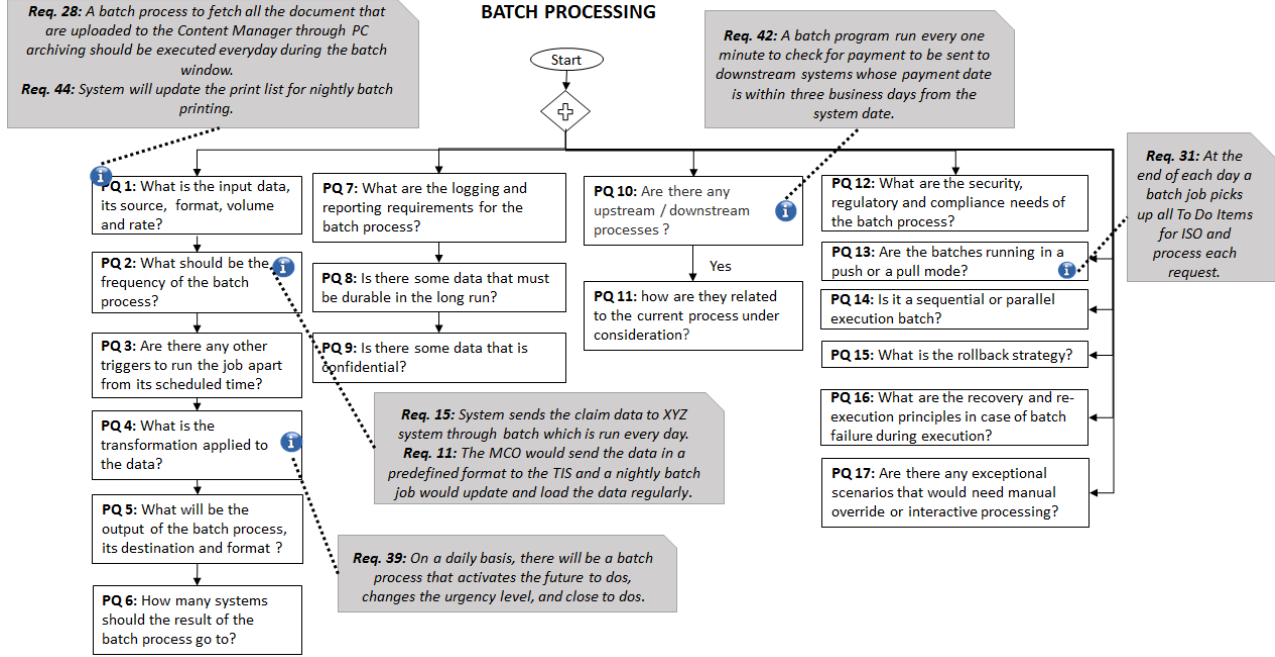


Figure 3: Annotated PQ-flow for Batch Processing

Table 4: Accuracy by which relevant PQ-flows are identified

Category	A	B	C	Prec.	Recall	F-Score	Spec.
Audit Trail	35	33	32	0.97	0.91	0.98	0.94
Batch Proc.	28	29	28	0.97	1.00	0.99	0.98
BPSA	59	53	52	0.98	0.88	0.97	0.93
Report	25	24	23	0.96	0.92	0.99	0.94
Workflow	14	12	11	0.92	0.79	0.99	0.85
AVERAGE				0.96	0.90	0.98	0.92

A – Relevant SRSs, B-Retrieved SRSs, C- Retrieved and Relevant SRS

4.3 ANNOTATING THE PQ-FLOWS

Once relevant PQ-flows are identified by the relevance algorithm, the BA can follow the flow of questions in order to elicit architecturally significant information. However, a naïve use of the PQ-flow could lead the BA to asking questions which are already addressed in the SRS. In this phase of our approach we therefore leverage data mining techniques to identify existing requirements which could be associated with various nodes in the PQ-flow. For illustrative purposes, in Figure 3, we present the Batch Processing PQ-flow with requirements automatically retrieved from one of the SRS and associated with a specific PQ. Some of the example requirements in Figure 3 are obfuscated for confidentiality reasons.

In this section we describe the techniques we used to associate requirements with specific PQs. It is worth noting that this task is even more challenging than classifying ASFRs by category. In effect, it represents an even more fine-grained task in which specific ASFRs are associated with individual PQs in the PQ-flows. For this task, we used the standard Vector Space Model (VSM) [29] to compute the similarity between a PQ and each requirement in a given SRS. The process is repeated for all the PQs in a PQ-flow for each ASFR category.

For similarity score computation, VSM uses the following formula:

$$Sim(ASFR_i, PQ) = \frac{ASFR_i \cdot PQ}{\|ASFR_i\| \|PQ\|} = \frac{\sum_{j=1}^N w_{j,a} w_{j,p}}{\sqrt{\sum_{j=1}^N w_{j,a}^2} \sqrt{\sum_{j=1}^N w_{j,p}^2}}$$

Where, ‘N’ represents the total number of unique words, ‘a’ represents ASFR statement, ‘p’ represents probing question, $w_{j,a}$ represents the list of frequencies of words present in i^{th} index of the ASFR statement and $w_{j,p}$ represents the list of frequencies of words present in PQs.

4.3.1 Analysis of Results for Annotating PQ-flows

To evaluate the results, three of the researchers (PRA, BB, AS)⁴ manually analyzed all possible associations (between PQ and ASFR statements). Each researcher individually reviewed all the associations. We periodically reviewed each other’s work to ensure ongoing agreement about the association. The resulting dataset formed the ‘answer set’ against which the association results could be compared. The task took a total of 80 hours to complete. Results are reported in Table 5 using the standard metrics: Precision, Recall, F-score, and Specificity. On an average, the precision was 0.28 while the recall was above 0.70 indicating that our approach was able to retrieve many of the relevant statements for annotation.

The lowest recall was observed in case of BPSA (0.58). For BPSA, we observed that VSM picked up comparatively more irrelevant terms for similarity matching. Further, the PQs we used for our experiments were taken verbatim from the SAs. However, our analysis revealed that rephrasing the PQs could improve the results further. For example, a PQ for BPSA “What are the roles that can configure the categories/types?”, if rephrased to make it more precise, such as “Who are the stakeholders (roles) that can configure the categories/types of the alert message?” would help in retrieving more relevant statements. This can be a direction for our future work.

⁴ PRA – P.R.Anish, BB – B.Balasubramaniam, AS – A.Sainani

Table 5. Results of associating requirements with PQ-flows

Category	#PQs	#ASFR Count	Potential associations	Relevant associations	Retrieved associations	Retrieved & relevant associations	Precision	Recall	F-score	Specificity
Audit Trail	16	89	1376	22	68	18	0.27	0.82	0.40	0.96
Batch Proc.	15	90	1350	207	674	162	0.24	0.78	0.37	0.55
BPSA	12	246	2952	90	434	52	0.12	0.58	0.20	0.87
Report	22	90	1980	63	609	49	0.08	0.78	0.15	0.71
Workflow	18	84	1512	52	59	31	0.58	0.65	0.61	0.98

We observed an average specificity above 0.80 indicating that the approach successfully discarded a significantly large number of irrelevant statements. We note here that in the case of our dataset, the ratio of the total number of possible annotations to the total number of relevant annotations was approximately 20:1 thereby making the annotation task very challenging.

5. EVALUATING PQ-FLOWS

We performed two studies to evaluate the usefulness of the approach in situ. Each study involved a BA, a pseudo-customer and a SA. In both studies, the BA and SA were experienced team members responsible for delivering large projects in the Insurance and Banking domains. The pseudo-customer had more than 20 years of experience working closely with customers to understand their requirements and to architect systems. Study 1 involved the SRS document from an ongoing insurance project containing a total of 226 requirements, while Study 2 was based on the SRS from an ongoing banking project containing a total of 2855 requirements. We ran the Relevance algorithm for each of the five ASFR categories against the two SRSs to find out the ASFR categories relevant to that SRS. For the SRS in Study 1, the relevance algorithm identified 4 requirements of *Audit Trail* and 2 requirements of *Business Process "State" alert*. For the SRS in Study 2, the relevance algorithm identified 17 requirements of *Report*, 22 of *Business Process "State" alert*, 1 of *Workflow* and 6 of *Batch Processing*. We then shared the relevant ASFR categories (their name, definition and an example) with the participants of Study 1 and Study 2 respectively and asked them to select the category they were most experienced and comfortable with. The participants of Study 1 chose *Audit Trail* while the participants of Study 2 chose *Report*.

In each of the studies, we ran VSM to analyze the respective SRS in order to generate annotated PQ-flows for *Audit Trail* and *Report* (as explained in section 4.3). For *Audit Trail*, VSM annotated seven of the 16 PQs while for *Report*, VSM annotated 21 of the 22 PQs. We manually analyzed these annotations and found the following:

For the SRS in Study 1 (*Audit Trail*), seven of the PQs were annotated. Of these, PQs were fully answered in two cases, while meaningful information was provided for three additional PQs which enabled better or more informed questions to be asked. The remaining two cases represented false positive annotations. Further, we manually analyzed the SRS and found that there was no additional relevant information in the SRS that should have been used to annotate the PQs.

For the SRS in Study 2 (*Report*), out of the 21 annotated PQs, three PQs were fully answered, while meaningful information, which enabled better or more informed questions to be asked, was provided for two additional PQs. The remaining 16 cases represented PQs with false positive annotations. These results are

not surprising given the relatively low precision reported for *Report* annotations in Table 5. Further, we manually analyzed the SRS and found that there were two additional requirements in the SRS that should have been included in the annotation but were excluded.

We provided the automatically annotated PQ-flows to the respective BAs of the two studies. For each study, the respective BA was asked to follow the PQ-flow and accordingly ask probing questions to the pseudo-customer and record the customer's response. In the flow, if a PQ was annotated (indicating that either an answer to that PQ is already present in the SRS or the SRS contains some information related to that PQ), the BA was asked to review the annotation and decide whether to skip that PQ (in case it completely answers the PQ) or ask a better or more informed question (in case the SRS contains some related information).

Based on the responses received from the pseudo-customer, the BA was asked to augment the original SRS with the answers received for the PQs. For *Audit Trail*, 13 new requirements were added and for *Report*, 15 new requirements were added.

After the completion of this activity, in each study, the BA was sent a questionnaire to gather feedback on the applicability and usefulness of the PQ-flow. Further, we shared the original and the augmented version of the SRS with the SA and asked them to evaluate the usefulness of the additional requirements. We summarize part of the feedback below. The questionnaire for the BA and the SA can be viewed at our project page <http://re.cs.depaul.edu/ASFR>

It should be noted that (1) two of the authors (PRA and SG) explained the approach and the evaluation study process to the study participants but none of the authors were present when the actual evaluation study was conducted. (2) the participants for the evaluation study were selected on the basis of their domain knowledge and project management experience endorsed by their supervisors who recommended them to us.

5.1 Feedback from the BA

To the question on the applicability of the PQ-flows in the requirements elicitation process, the BA from Study 1 stated: "It (the PQ-flow) fits well into the elicitation process as it covered the key architectural concerns that a software architect will have. I as a BA am not able to elicit them on my own due to lack of expertise in the architectural domain. So this is a really good approach". To the question on the usefulness of the hierarchical structure of the PQ-flows (versus just having a collection), the BA from Study 1 stated: "Yes, having a hierarchical structure really helps in specifying each problem area. For example, having a flow structure to the data audit aspect helps clarify in stages, the volume of the data, its type and output. Having it as a collection instead could possibly lead to several overlapping or confounding

responses which could potentially require follow-up questions” and the BA from Study 2 said “PQs that depended on other PQs were well represented in hierarchical structure. This helped in traversing to the right question without wasting much of time in thinking what to ask next?” To the question on the extent to which the BA could imagine integrating the information acquired through using the PQ-flows into the SRS, the BA from Study 2 said: “I would think that a good fit would exist between the information gained in the PQ-flows and the requirements specification. It would make the requirements specification much richer in terms of architectural knowledge” and when asked whether they would use PQ-flows in future projects, the BA from Study 1 said: “PQ-flows offer a sound structured framework to collect information and therefore I would definitely like to use this approach in future projects” and the BA from Study 2 said “I would think that this approach would apply to any data-intensive project, which is any typical project. PQ Flows offers a sound structured framework to collect information”. Further, on the question on the change in the order the BA would like to see to make the PQs more helpful to them, both the BAs responded saying that the current order of the PQs is “good enough”.

5.2 Feedback from the SA

To the question on the value seen by the SA on using this approach, the SA from Study 1 said: “I think this approach is quite exhaustive to make customer as well as vendor think in depth before requirements are put forth and are understood clearly. It would avoid the SA make any assumptions about unstated or unclear requirements and also will save us a lot of time that gets wasted going back and forth to the customer for clarifications”. The SA from Study 2 said: “I think, this approach shall help us to ensure that the customer does not miss out key features that he may not think otherwise. The flow shall serve as a template for similar requirements. The customer shall view us as a good process oriented and experienced organization”. When asked about the kind of effort needed to get this level of architectural detail in the absence of such an approach, the SA from Study 1 said: “Effort would be quite high and more than that user acceptance testing will raise questions as to why things were not clarified earlier in requirements phase. Series of change requests will follow after implementation.” Further, the SA from Study 2 mentioned: “Yes, this approach will help very much in bridging the gaps that exists between requirements and architecture”. On the question “Do you feel a BA equipped to ask these questions is useful or would you prefer asking these questions on your own”, the SA from Study 1 said “the BA would need some amount of training which looks very much doable to me. Once trained, this approach would definitely be very useful and would save a lot of time which we spend going back to the BA or client for clarifications”. The SA from Study 2 said “I am not sure the knowledge BA will have in terms of domain, overall understanding of customer process flow as well as the organizational structure with roles and responsibilities. If well equipped with this then can carry out such exercise and would be very useful. An important aspect that is required in an individual who does this job is to do lateral thinking and should be analytical in nature”. Further when asked, if such an approach helps in bridging the gap that exist between requirements and architecture, the SA from Study 1 said “Yes, very much!!”

6. THREATS TO VALIDITY

We used multiple research techniques (i.e. interviews and online survey) for carrying out the ground work that formed the basis for contextualizing PQ-flows. We evaluated the threats to validity in

this multi-method research by using the checklist in [15]. As our design of PQ-flows is exploratory in nature and relies on refining the PQ-flows at each phase by collecting practitioners’ feedback, the generalizability of our PQ-flow structures is our most important concern. To what extent can the observations of the interview study participants and the two surveys’ participants be considered representative for other project organizations? We cannot claim universal generalizability of the PQ-flows that we derived by using our participants’ responses. However, it is reasonable to expect similar experiences in projects and organizations working in the application domains in which our practitioners work. For example, other organizations in sectors that must ensure compliance of their IT-solutions to business sector-specific regulations might have observations similar to those in this case study. More research is needed to collect evidence for or against this generalization.

We are also aware of the inherent uncertainty of interview techniques and of survey methods [15]. The first uncertainty is about the extent to which the participant’s answered our questions truthfully. As in [3], we minimized this uncertainty by involving volunteers, so that they respond to our survey out of their free will and without pressure. Second, the researchers might have included their bias in the data collection and analysis process. To reduce this threat, we used Yin’s recommendations [30]. For example, we ensured that our participants were drawn from diverse application domains, which let us evaluate the PQ-flows from diverse perspectives. Additionally, we had the data analysis report reviewed by some interviewees (some of whom provided feedback on clarifying the concepts, but this did not change the analysis).

Furthermore, regarding the evaluation of the classification techniques chosen in the tool design, we used a large set of SRSs (114), comprising over 8000 statements. Although this cannot serve for forming universal claims, it injects realism [31] in our analysis as the SRSs have been delivered in large real-life industrial projects with elaborate complex functionalities delivered by team sizes greater than 100 to hundreds of thousands of users.

Finally, the feedback from the BA and SA about the utility of our approach could be ascribed to the ‘good participant role’ in which the participant attempts to discern the experimenter’s hypotheses and to confirm them [52]. The participant does not want to ‘ruin’ the experiment. This threat cannot be eliminated, but our future empirical evaluations should provide better evidence on the utility of our approach.

7. RELATED WORK

Much work has been performed at the intersection of requirements engineering and software architecture design [32-40]. In 2001, Nuseibeh proposed the Twin Peaks model to depict the interdependencies that exist between the problem and solution space – often referred to as requirements and architectural design [10]. Since requirements specifications and architecture design affect and constrain each other, this model emphasized the need to progressively discover and specify requirements while concurrently exploring alternative architectural decisions. Since then, many researchers have focused their efforts on understanding the impact of requirements on architecture [42-44]. There is also active work that investigates traceability from requirements to architectural designs [45, 46], including links between key stakeholder concerns, architecturally significant requirements, important architectural decisions and sections of code where architectural decisions are implemented [45]. Goknil

et al., [46] used the semantics of traces, requirements relations and architecture verification techniques to automatically generate and validate trace links between requirements and architecture leveraging model transformation in ATL and term-rewriting logic in Maude.

Most work in the area of requirements and architecture is restricted to NFRs and their impact on architecture [5, 47, 48]. Chen et al., [47] leveraged the relationship between Architecturally Significant Requirements and Architectural Design Decisions to co-develop NFRs and architecture while Niu et al. [5] introduced a practitioner-oriented approach to analyze and evaluate architecturally significant requirements for enterprise systems.

Researchers have also addressed the problem of missing requirements [8, 13]. Poort et al., [8], proposed a facilitated method, PALM, which elicits business goals and establishes the link between those goals and the quality attribute requirements for a system under development. It helps to discover missing quality attributes and empowers the architect to question the necessity of overly stringent requirements by appealing to stakeholder-expressed business goals. Salehin [13] investigated the extent to which requirements information is missing during the software architecture process and the impact of that missing information on system architecture in terms of effort. They found that SAs actively search for strategies to reduce missing information in the requirements, so that the software architecture is designed with less effort. This highlights the importance and the need for PQs as a mechanism to unearth unstated architecturally relevant information from ASFRs. Chen et al. [48] presented an evidence-based framework for systematically characterizing architecturally significant requirements. In particular, their finding that architecturally significant requirements tend to be described vaguely contributed to the motivation for our work on PQ-flows.

Many techniques for identification, classification and mining various types of requirements can be found in the literature [28]. However, the focus has primarily been on NFRs. Based on the expert interviews, we have confirmed that many requirements which are purely functional in nature have architectural impact and we focus on such requirements. To the best of our knowledge, this is the first time the concept of PQ-flows and their contextualization is introduced.

8. CONCLUSION

This paper presents results from a study with experienced SAs to identify, document, and organize PQs for five different areas of functionality into structured flows. It makes two contributions. The first is a set of PQ-flows aimed at empowering BAs to ask questions that unearth details needed by SAs to make informed architectural decisions. The second is the process of deriving these PQ-flows, which contributes to the body of knowledge for relating RE and SA. We deploy machine learning techniques to contextualize the use of the PQ-flows and to guide BAs through the process of asking pertinent questions that elicit architecturally significant details. The BAs can leverage this information to determine what is already known, what knowledge is missing, and what effect the missing knowledge would have on the flow of the PQs. Results from our initial study showed that the use of PQ-flows by BAs helped to unearth many architectural details and led to more complete and comprehensive SRSs enriched with architectural information. The study also suggested BAs would need initial training to effectively use this approach.

The work has implications for practice and research. First, we found an inexpensive down-to-earth solution that seems effective for enriching SRSs with architecturally relevant information. Using this approach, BAs can produce richer specifications that contain the details that SAs need for making architectural decisions. This could lead to less communication cycles between SAs and customers as there would be little need for follow-up clarifications in later project stages concerning ASFRs. Second, our approach can facilitate creation of a repository of PQ-flows for each ASFR category which could generate a body of knowledge similar to the Design Patterns of Gamma et al. [14] which could be used by the BAs to enhance the quality and completeness of SRSs.

Future research directions could involve more rigorous studies of PQ-flows in practice, and exploring the use of PQ-flows for emergent domains such as Internet Of Things wherein the architectural landscape is still evolving and regulations are nascent at best. We are in the process of conducting experiments on larger and more varied datasets to evaluate the scalability of our approach and to improve its accuracy. Based on our current experiments, the training process took 140 hours, but once trained, it took us 13-15 minutes to process a large SRS containing 8669 individual requirements. We plan to further address performance as we transition our approach to industrial practice.

9. ACKNOWLEDGEMENTS

The authors thank all study participants. Co-author Cleland-Huang's participation was funded by United States National Science Foundation Grant # CCF-0810924.

10. APPENDIX

For repeatability purposes, we document the parameter values used in our experiments. Definitions for each parameter can be found in appropriate documentation manuals (for example, the Weka Documentation Manual [50]).

For All Experiments

Pre-processing

StringToWordVector

```
weka.filters.unsupervised.attribute.StringToWordVector -R 1 -W
10000 -prune-rate -1.0 -C -N 0 -L -S -stemmer
weka.core.stemmers.LovinsStemmer -M 1 -tokenizer
weka.core.tokenizers.WordTokenizer -delimiters \"
\\r\\n\\t.,;:\\\\|\\\"'()?!\"
```

Data type conversion

```
weka.filters.unsupervised.attribute.NumericToNominal -R first-last
```

Attribute Selection

InfoGain

```
weka.filters.supervised.attribute.AttributeSelection -E
weka.attributeSelection.InfoGainAttributeEval -S
weka.attributeSelection.Ranker -N 100
```

Binary Classification

Subsampling (Spread Subsampling)

```
weka.filters.supervised.instance.SpreadSubsample -M 1.0 -X 1147.0
-S 5
```

Oversampling (SMOTE)

```
weka.filters.supervised.instance.SMOTE -C 1 -K 5 -P 70.0 -S 1
```

Classification (Naïve Bayes)

```
weka.classifiers.bayes.NaiveBayes
useKernelEstimator = False, useSupervisedDiscretization = False
```

Multi-label Classification (RaKEL)

```
mulan.classifier.meta.RaKEL; NumModels = 0, SizeOfSubset = 3
```

11. REFERENCES

- [1] T. Tuunanen and M. Rossi, "Engineering a Method for Wide Audience Requirements Elicitation and Integrating It to Software Development," in *Proceedings of 37th Hawaii Int. Conference on System Sciences*, Big Island, Hawaii, USA, 2004, IEEE, 10
- [2] P.R. Anish, B. Balasubramaniam, J. Cleland-Huang, R. Wieringa, M. Daneva, S. Ghaisas. Identifying Architecturally Significant Functional Requirements, *TwinPeaks 2015*. IEEE Press, 3-8.
- [3] P.R. Anish, M. Daneva, J. Cleland-Huang, R. Wieringa, S. Ghaisas. What You Ask Is What You Get: Understanding Architecturally Significant Functional Requirements, *RE 2015*, IEEE Press, 86-95.
- [4] M. Glinz. A Glossary of Requirements Engineering Terminology. International Requirements Engineering Board (2011).
- [5] N. Niu, L.D. Xu, J.C. Cheng, Z. Niu. Analysis of Architecturally Significant Requirements for Enterprise Systems, *IEEE Systems Journal*, 8(3), 850-857
- [6] A. Koziolok. Architecture-driven Quality Requirements Prioritization, *TwinPeaks 2012*, IEEE Press, 15-19
- [7] D. Ameller, C. Ayala, J. Cabot, X. Franch. Non-functional Requirements in Software Architecture Practice, report ESSI-TR-12-1, Universitat Politècnica de Catalunya, Barcelona, 2012; <http://hdl.handle.net/2117/15716>
- [8] E. Poort, N. Martens, I. van de Weerd, H. van Vliet. How Architects see Non-Functional Requirements: Beware of Modifiability, *REFSQ*, 2012, 37-51
- [9] Cleland-Huang, J., Czauderna, A., & Keenan, Ed., A persona-based approach for exploring architecturally significant requirements in agile projects, *REFSQ '13*, 18-33
- [10] N. A. Qureshi, M.B. Usman, N. Ikram. Evidence in Software Architecture, a systematic literature review, *EASE 2013*, 97-106
- [11] M. Galster, M. Mirakhorli, J. Cleland-Huang, J. E. Burge, X. Franch, R. Roshandel, and P. Avgeriou. Views on Software Engineering from the Twin Peaks of Requirements and Architecture. *SIGSOFT Softw. Eng. Notes* 38, 5, 2013, 40-42.
- [12] J. Cleland-Huang, R.S. Hanmer, S. Supakkul, and M. Mirakhorli. 2013. The Twin Peaks of Requirements and Architecture. *IEEE Softw.* 30, 2, 24-29.
- [13] Z. Li, P. Liang, P. Paris Avgeriou, Application of Knowledge-based Approaches in Software Architecture: A Systematic Mapping Study, *Information & Software Technology*, 55, 2013, 777-794
- [14] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*, Addison-Wesley: Reading, MA, 1995
- [15] R.J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014
- [16] Dillman, D.A., Smyth J.D., Christian, L.M., *Tailored Design Method Hardcover*, Wiley, 2014
- [17] M.B. Miles, A.M. Huberman, *Qualitative Data Analysis*, 2nd ed., Sage Publications, Thousand Oaks, CA, 1994
- [18] C. Charmaz, *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*, Sage, Thousand Oaks, 2006.
- [19] G. H. John, P. Langley. Estimating Continuous Distributions in Bayesian Classifiers. 11th conference on uncertainty in Artificial Intelligence, San Mateo, 338-345, 1995
- [20] N. V. Chawla. Data Mining for Imbalanced Datasets: An Overview. In. Maimon O., Rokach L. (eds.) *The data mining and knowledge discovery handbook*, Springer 2005, 853-867
- [21] J.B. Lovins. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics*, vol.11, nos.1 and 2, March and June 1968
- [22] M. A. Hall, G. Holmes. Benchmarking Attribute Selection Techniques for Discrete Class Data Mining, *IEEE Transactions on Knowledge and Data Engineering*, vol.15, no.3, May/June 2003
- [23] G. Tsoumakas, I. Katakis, I. Vlahavas. Random k-labelsets for Multilabel Classification. *Knowledge and Data Engineering, IEEE Transactions on*, 23(7), pp. 1079-1089, 2011.
- [24] M. Zhang, Z. Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition* 40(7):2038—2048, 2007
- [25] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, I. Vlahavas. *Mulan: A Java Library for Multi-label Learning*. *The Journal of Machine Learning Research*, 12, 2411-2414, 2011
- [26] W.B. Frakes, R. A. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Englewood Cliffs, 1992
- [27] Altman DG, Bland JM (1994) Statistics notes: diagnostic tests 1: sensitivity and specificity. *Br Med J* 308(1552).
- [28] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc. Automated Classification of Non-functional Requirements. *Requirement Engineering* 12(2): 103-120 (2007)
- [29] G. Salton, A. Wong, C. S. Yang, A vector space model for automatic indexing, *Communications of the ACM*, v.18 n.11, p.613-620, Nov. 1975.
- [30] R.K. Yin, *Case study research*, Sage, 2014
- [31] Wieringa, R.J. and Daneva, M. (2015). Six strategies for generalizing software engineering theories. *Science of computer programming*, 101(April 2015), 136-152
- [32] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed., Reading, MA: Addison-Wesley, 2011.
- [33] D.M. Dikel, D. Kane, and J.R. Wilson, *Software Architecture: Organizational Principles and Patterns*, Upper Saddle River, NJ: Prentice-Hall, 2001.
- [34] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*, Boston: Addison-Wesley, 2000.
- [35] A. Ran, *ARES Conceptual Framework for Software Architecture*, in M. Jazayeri, A. Ran, and F. van der Linden, ed., *Software Architecture for Product Families Principles and Practice*, Boston: Addison-Wesley, 2000, 1-29.
- [36] M. A. Babar, A. W. Brown, and I. Mistrik. *Agile Software Architecture: Aligning Agile Processes and Software Architectures*. Morgan Kaufmann. 2013.
- [37] J. Bosch, *Design and Use of Software Architecture: Adopting and Evolving a Product-Line Approach*, Boston: Addison-Wesley, 2000.
- [38] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, *Documenting Software Architectures: Views and Beyond*, Boston: Addison-Wesley, 2002.
- [39] P. Clements, L. Northrop, *Software Product Lines: Practice and Patterns*, Boston: Addison-Wesley, 2002.
- [40] J. Garland, R. Anthony, *Large-Scale Software Architecture: A Practical Guide using UML*, Wiley, 2002
- [41] B. Nuseibeh, *Weaving Together Requirements and Architectures*. *Computer*, 2001. 34(3): 115-117

- [42] M. Galster, M. Mirakhorli, J. Cleland-Huang, J. E. Burge, X. Franch, R. Roshandel, and P. Avgeriou. Views on Software Engineering from the Twin Peaks of Requirements and Architecture. *SIGSOFT Softw. Eng. Notes* 38, 5, 2013, 40-42.
- [43] J. Cleland-Huang, R.S. Hanmer, S. Supakkul, and M. Mirakhorli. 2013. The Twin Peaks of Requirements and Architecture. *IEEE Softw.* 30, 2, 24-29.
- [44] Gross, A., Doerr, J., What do software architects expect from requirements specifications? Results of initial explorative studies, *Twin Peaks 2012*, pp.41-45.
- [45] J. Cleland-Huang, Thinking about Quoins: Strategic Traceability of Architecturally Significant Requirements, *IEEE Software*, vol. 31, no. 4, September/October 2013, 16–18.
- [46] A. Goknil, I. Kurtev, K. Van Den Berg, (2014), Generation and validation of traces between requirements and architecture based on formal trace semantics. *Journal of Systems and Software*, 88, 112-137
- [47] Chen, F. (2014, August). From architecture to requirements: Relating requirements and architecture for better Requirements Engineering., *RE IEEE Press*, 451-455.
- [48] L. Chen, M. Ali Babar, and B. Nuseibeh, Characterizing Architecturally Significant Requirements, *IEEE Software*, vol. 30, no. 2, pp. 38–45, 2013.
- [49] G. Marek, C. Adam and J. Cleland-Huang, Towards mining replacement queries for hard-to-retrieve traces, *Proceedings of the IEEE/ACM ASE 2010*, 245-254.
- [50] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2 edition, 2005.
- [51] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.
- [52] A.L. Nichols, J.K. Maner, The good subject effect: Investigating participant demand characteristics, *Journal of General Psychology*, 135, 151-165, 2008