Identifying Architecturally Significant Functional Requirements

Preethu Rose Anish[1], Balaji Balasubramaniam [1], Jane Cleland-Huang[2], Roel Wieringa[3], Maya Daneva[3], Smita Ghaisas[1]
TATA Research Development and Design Centre (TRDDC) [1],
TATA Consultancy Services Ltd., Pune, India
{preethu.rose, balaji.balasubramaniam, smita.ghaisas}@tcs.com
Systems and Requirements Engineering Center (SAREC) [2] DePaul University, Chicago USA
{jhuang}@cs.depaul.edu
University of Twente[3], Enschede, the Netherlands
{r.j.wieriga, m.daneva}@utwente.nl

***Abstract*—Failure to identify and analyze architecturally significant functional and non-functional requirements (NFRs) early on in the life cycle of a project can result in costly rework in later stages of software development. While NFRs indicate an explicit architectural impact, the impact that functional requirements may have on architecture is often implicit. The skills needed for capturing functional requirements are different than those needed for making architectural decisions. As a result, these two activities are often conducted by different teams in a project. Therefore it becomes necessary to integrate the knowledge gathered by people with different expertise to make informed architectural decisions. We present a study to bring out that functional requirements often have implicit architectural impact and do not always contain comprehensive information to aid architectural decisions. Further, we present our initial work on automating the identification of architecturally significant functional requirements from requirements documents and their classification into categories based on the different kinds of architectural impact they can have. We believe this to be a crucial precursor for recommending specific design decisions. We envisage ArcheR, a tool that (a) automates the identification of architecturally significant functional requirements from requirement specification documents, (b) classify them into categories based on the different kinds of architectural impact they can have, (c) recommend probing questions the business analyst should ask in order to produce a more complete requirements specification, and (d) recommend possible architectural solutions in response to the architectural impact.**

***Index Terms*—Architecturally Significant Functional Requirements (ASFRs), architectural decisions, requirements classification**

I. Introduction

Requirements Engineering (RE) activities involve the capture of both functional and non-functional requirements (NFRs[1]), respectively describing what the system needs to do and how it is to be achieved. Functional requirements (FRs) are collected and documented by business analysts and domain experts. Architectural decisions, on the other hand, are made by technology experts. The knowledge corpora for FRs and architectural solutions are therefore maintained separately. Almost all FRs may be argued to have some extent of impact on architecture. However, our focus is on those FRs that have *significant* impact on architecture. By this, we mean any FR that is critical, is of high-risk, is volatile, involves expensive-refactoring or has legislative impact. We refer to such requirements as 'Architecturally Significant Functional Requirements' (ASFRs). For instance, a FR statements such as *Ability to receive notification whenever a transaction is made* carry an architectural impact. The kind of impact they can have on architecture (e.g. *email notification or real time notification with an ability to subscribe to topics of interest*) is typically not explicitly stated in the FR statement. Also, the actual architectural decision meant to address the impact and its rationale is not explicitly connected to the FR. For example *for simple email, an event-driven architecture would suffice; for real time notification with the ability to subscribe to different topics, publish-subscribe* – a different architectural style is required [14]. Some tools, such as IBM's Rational DOORS, are helpful for establishing traceability between FRs and design items of architectural significance, but the traceability links are created manually during the design process [16]. ASFRs can be difficult to identify from a given set of FRs. The business analysts do not have the requisite technical knowledge to infer and articulate the architectural impact from the FRs that they capture from customers. As a consequence, ASFRs carry hidden architectural implications that need to be unearthed. This scenario is quite different from NFRs that explicitly solicit architectural considerations. In fact, NFRs or quality requirements are considered the main drivers for architectural decisions [19, 20, 25]. For example, a NFR such as *the system shall not be unavailable more than 1 hour per 1000 hours of operation* is clearly an availability requirement with an explicit architectural impact. On the other hand, an ASFR such as *Ability to receive notification whenever a transaction is made* has implicit architectural impact and the design solution would change based on the 'notification type'. Since the type of desired notification is not stated in the FR, it may lead to wrong architectural decisions. To remedy this situation, we envisage a situation where business analysts are equipped to identify ASFRs and would be able to understand some of their key implementation decisions. In this situation they can ask additional relevant questions during the requirements gathering process and produce a more complete requirement specification. We term the questions that aid

[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.

978-1-4673-7100-1/15 $31.00 © 2015 IEEE
DOI 10.1109/TwinPeaks.2015.9

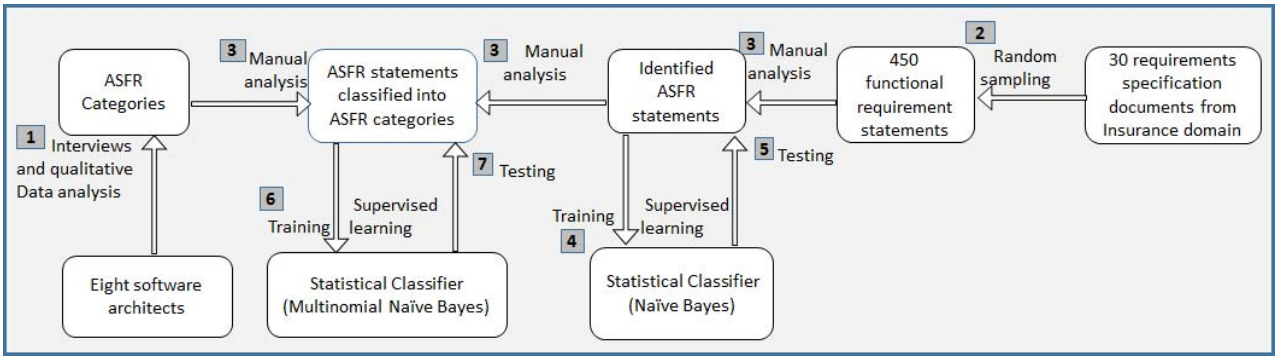

Fig. 1. Research methodology. (The numbers indicate the sequence of research activities)

architectural decision making 'Probing Questions (PQs)'. Much of the prior literature [9, 24] on architecturally significant requirements has focused on quality concerns related to e.g. security or reliability. In contrast to this, we focus on FRs. The ASFRs may not specifically highlight quality concerns, but nevertheless strongly impact architectural decisions. To the best of our knowledge, this is the first attempt to detect and classify ASFRs automatically.

To develop an understanding of how software architects cope with ASFRs in real projects, we conducted semi-structured interviews with experienced architects from a large IT organization in India. We used open-ended questions. The main objective of our interviews was to qualitatively explore if the architects have encountered ASFRs in their projects; and if so, how they perceive, interpret (including PQs), represent and finally, link them to architectural decisions.

Based on the interview findings, we envisage ArcheR, a tool that (a) automates the identification of ASFRs from requirement specification documents, (b) classifies them into categories based on the different kinds of architectural impact they can have, (c) recommends PQs the business analyst should ask in order to produce a more complete requirements specification, and (d) recommends possible architectural solutions in response to the architectural impact. In this paper, we present our initial results for (a) and (b).

The rest of the paper is organized as follows: Section II details the empirical study and presents our early results on automating the identification and classification of ASFRs, section III details the related work, section IV presents threats to validity while section V presents discussion and conclusion.

II. Architecturally Significant Functional Requirements (ASFRS)

This section details (a) the empirical study we conducted with the aim to better understand ASFRs from practitioner's perspective and, (b) our results on automating the identification and classification of ASFRs. Figure 1 depicts our research methodology.

A. Empirical Study for Understanding ASFRs

For the purpose of our empirical study, we chose a qualitative interview-based research method. As compared to other qualitative research strategies such as structured questionnaires, it often provides a more flexible approach by letting us better investigate interesting issues that appear during conversations [28]. We interviewed 8 experienced architects covering a variety of application domains including Banking, Insurance, Manufacturing, Embedded systems and Medical. The experience of the architects was in the range of 9 to 22 years. Table I summarizes the participants' details.

TABLE I. Interview Participant Details

Participant ID	Experience (yrs.)	Application Domain
P1	9	Medical
P2	15	Banking
P3	14	Insurance
P4	22	Embedded Systems
P5	19	Insurance
P6	20	Insurance
P7	19	Banking
P8	11	Manufacturing

Our data analysis was guided by the reasoning that underlies the sense-making techniques associated with the less procedural versions of the Grounded Theory (GT) [17]. Specifically, we applied the techniques of coding and constant comparison recommended by Charmaz [28]. This approach is exploratory and well suited for situations where the researcher does not have pre-conceived ideas. Our choice of using GT for data analysis agrees with Matavire and Brown [29] who profiled the use of GT in information system research.

The analysis of the interview transcripts revealed that all the 8 architects emphasized that most of the times the architectural impact in functional requirements is implicit and the impact is unearthed only after a series of follow up questions are asked to the customer. Based on our data analysis using the coding techniques from GT, we developed an initial classification for ASFRs and a set of PQs pertinent to each ASFR category. Further, the questionnaire also included questions pertinent to architectural solution / decisions that the architect took. These questions would be a crucial input for our planned work on the recommendation system.

TABLE II. ASFR CATEGORIES AND EXAMPLES

ASFR Category	Description	Example ASFR	Sample Probing Question(s)
Audit trail	Facilitate auditing of system execution	The system must record every modification to customer records for audit purposes.	- Do we need to capture pre and post images of database tables? -What kind of logs do we need to maintain?
Batch processing	Facilitate batch processing	The disbursement process should be a daily batch process in xxx for the regular claim pay-outs.	-What is the likely processing load? -Who all needs to be notified after the process is complete?
Localization	Provide support for multiple languages	During the Term, the Authority may require the Contractor to deliver specific communications in other languages	- Are there regulatory/ legal requirements in that region that necessitate a particular kind of delivery of content? - What type of language standardization do we have to use? -Whether localization should be triggered automatically or should be on a manual basis?-
Communication	Provide support for communication	Workflow is required to send notification to the Underwriters	- Should the communication be configurable by customer? -Should the communication be uni-directional or bi-directional? -What mode of communication is needed?
Payment	Facilitate financial transactions	The Contractor shall return the excess premium payment to xxx through the interim or annual claim process	- Payment through debit card or credit card or net banking or monthly direct debit? - Link payment with one bank or with multiple banks? - Should payment details to be transferred to some other system?
Print	Provide support for printing documents	The Commission Statement should be printed using the package-supplied format.	- Is the application running on virtual environment? - Print locally / centrally? - Print driver availability?
Report	Facilitate report generation	System should generate reports on the Complaint Register on a monthly basis.	- Type and filters for the report? - Option to export as pdf, excel? - Is report customization facility needed? - Report viewing capability. Whether seen by all or set of people? - Real time reporting or end of the day reporting?
Search	Provide support for search functionality	Claims Assessor should be able to search for the claims record to be processed.	- Based on keyword or based on filters? - Is semantic search needed? - Channel through which the end customer will search - search on mobile or search on computer?
Third party interaction	Facilitate interaction with third party components	Once an application has been applied for it will be exported to ABC Back office where it may be processed both automatically by the ABC Back office system and manually by an underwriter.	- What are the systems across which the application process spans? - Are there any ERP implementations in place with which we need a hand-shake? - Are there multiple systems with the same functionality as a result of mergers/acquisitions?
Workflow	Provide support to move work items, facilitate reviews and approvals.	All complaints should first hit work bin or a senior role assigned to it.	- Does it involve a set of approvals? - How quickly approval is needed? - How many levels of approval are needed? - Is versioning of workflow needed?
Online help	Facilitate online help	An online help facility should be available for the claim intimation process.	- Do you want this to be a feature based help? - Do you expect any case-based recommendations?
Licensing	Facilitate services for acquiring, installing, and monitoring license usage	There should be facility for installing and monitoring license usage.	- Is this an enterprise –wide license monitoring? - What is the mode of licensing? Is it pay per use?

However, we do not provide the details of architectural solutions here as it is out of scope for this paper. Table II presents the twelve categories of ASFRs identified by our study so far along with examples and PQs. It should be noted that we have obfuscated examples for confidentiality reasons. Our experts consistently flagged FRs of these categories as architecturally significant in nature. It should be noted that at this stage, we do not claim the exhaustiveness of these categories. We expect more categories to emerge as we expand the scope of our study.

B. Automated Identification and Classification of ASFRs

As our qualitative data analysis has shown that FRs can be architectural in nature and often have implicit architectural impact, we set out to develop a tool-ArcheR which is capable of flagging ASFRs and also of classifying them into specific ASFR categories. This is a necessary precursor for recommending specific PQs and design decisions. As an initial step in this direction, we present the details of our preliminary experiments in automating the identification and classification of ASFRs.

We report our results with Naïve Bayes Classifier (NB) as it out performed the other classifiers (Decision Tree, Support Vector Machine, Nearest Neighbor and Meta classifier – AdaBoost) in the case of our dataset and the optimization parameters chosen.

Our dataset comprises of 450 FR statements selected from 30 requirements specification documents (from the Insurance domain), by using the random sampling technique. As a part of our supervised learning mechanism, two of the industrial researchers manually analyzed the 450 FR statements to label each statement as an ASFR or a non-ASFR. Further, each ASFR was manually tagged into the categories listed in Table II. This formed the 'answer' set against which the identification and classification results could be compared. The manual identification and classification took approximately 7.5 hours of each person's time.

We used 10-fold cross-validation for training and testing. The dataset was divided into ten equal buckets. During each iteration, a single bucket was treated as test data and the classifier was trained on the remaining nine buckets. Ten such iterations were performed until each bucket had been tested.

1) Identifying Architecturally Significant Requirements from the given Set of FRs

In this phase the dataset (i.e. 450 FR statements) is classified into either an ASFR statement or a non-ASFR statement. We used WEKA data mining software [27] for pre-processing the statements. This involves removing stop words that do not provide any relevant information on the statement's lexical content (for example conjunctions and prepositions). The statements are then reduced into a set of keywords. The keywords are reduced to their stemmed form using Lovin's stemming algorithm [26], to reduce all the words with the same root (or, if prefixes are left untouched, the same stem) to a common form, usually by stripping each word of its derivational and inflectional suffixes. The resultant keywords form the feature attributes. We used Information gain [22] to rank these features and retrieve the top 40 feature attributes. Examples of top ranked feature attributes include *batch, search, communication, language, approval, print, email, workflow, audit, acknowledgment, payment.* These 40 feature attributes are then used as an input to train the NB classifier [21]. The NB probability model derived from these 40 attributes is then used for testing. The output of the testing phase involves flagging each FR statement as an ASFR or a non-ASFR. This training and testing mechanism is performed for the entire dataset by 10-fold cross validation. For ASFR identification, we achieved a precision of 77 % and a recall of 81 %.

2) ASFR Classification

For ASFR classification, the ASFRs obtained in section III B (1) are taken. The mechanisms for preprocessing, training and testing are the same as those explained in section III B above. We used Multinomial Naïve Bayes classifier for ASFR classification. Table III reports the two metrics of recall and precision for each ASFR category individually. Table IV depicts a confusion matrix for the classification results. The correct classifications (true positives) are depicted on the diagonal, and have been highlighted in the diagram. It should be noted that we have considered the original 450 requirements statement for precision and recall calculation. Also, the 450 FR statements did not contain ASFRs of category 'Online help' and 'Licensing'.

TABLE III. CLASSIFICATION RESULTS FOR EACH ASFR CATEGORY

ASFR Category	**Precision (%)**	**Recall (%)**
Audit trail	66.7	47.6
Batch processing	60.0	60.0
Localization	100	90.0
Communication	62.5	91.7
Payment	60.0	57.1
Print	75.0	50.0
Report	50.0	28.6
Search	57.9	52.4
Third party interaction	100	57.1
Workflow	78.9	65.2

Best results were achieved for 'Localization' category of requirements, for which we achieved 90% recall at 100% precision. Overall, recall ranged from 28% to 90% and precision from 50% to 100%. While analyzing the results, we found that 'Report' category had the lowest recall (28.6%) because we had very few statements in our dataset belonging to the category 'Report'. However, our prior work has shown that increasing the size of the training set can improve the accuracy of requirements classification results [9]. While there is room for improvement, these classification results provide solid initial evidence that the ASFR categories identified by our experts can be automatically classified. Furthermore, this provides the foundation needed for our planned recommendation activities.

TABLE IV. CONFUSION MATRIX DEPICTING CLASSIFICATION RESULTS

A	B	C	D	E	F	G	H	I	K	Total	←Classified as
10	0	0	6	1	0	0	3	0	1	21	A = Audit trail
0	**6**	0	0	2	0	0	2	0	0	10	B = Batch processing
0	0	**9**	1	0	0	0	0	0	0	10	C = Localization
0	0	0	**55**	2	2	0	1	0	0	60	D = Communication
1	2	0	4	**12**	1	0	0	0	1	21	E = Payment
0	0	0	8	2	**12**	2	0	0	0	24	F = Print
0	0	0	2	0	1	**2**	2	0	0	7	G = Report
3	2	0	3	0	0	0	**11**	0	2	21	H = Search
0	0	0	3	0	0	0	0	**4**	0	7	I = Third party interaction
1	0	0	6	1	0	0	0	0	**15**	23	J = Workflow

III. RELATED WORK

Over the last 20 years, extensive research has been conducted in the area of development and documentation of techniques, processes, guidelines, and best practices for software architecture design [1-4, 13, 30-33]. In 2001, Nuseibeh proposed the Twin Peaks model that draws attention to the synergistic relationship between requirements and architectural design [5]. This model emphasizes the need to progressively discover and specify requirements while concurrently exploring alternative architectural decisions. Since then many researchers have focused their efforts towards understanding the impact of requirements on architecture [6, 7, 34]. In [6], the authors conduct an exploratory study and report initial results on investigating information needs that should be fulfilled in software requirement specifications from the viewpoint of software architects. Chen et al. [8] present an evidence-based framework to systematically characterize architecturally significant requirements. In particular, their finding that architecturally significant requirements tend to be described vaguely, is in fact the motivation for this work. Cleland-Huang et al. [9] present an automated approach for classification of non-functional requirements concerns. In [10], the authors discuss trace links that can be established among key stakeholder concerns, architecturally significant requirements, important architectural decisions and sections of code where architectural decisions are implemented. Various models and related tools for capturing, managing and sharing architectural design decisions and their rationale exists [15]. In [11], the authors propose a method to assist software architects in architectural decision-making. They conceptualize the relationship between quality requirements and architectural decisions by using ontology to represent and manage architectural knowledge. In [12], the author's present ontology for architectural knowledge representation called Arteon.

IV. THREATS TO VALIDITY

The preliminary results reported in this paper are based on the analysis of 450 FR statements selected from the insurance domain. The training corpus cannot be held to be comprehensive at this stage. We anticipate variations in the results as we expand the scope of our study to cover more FRs from insurance domain and also from other domains. However, based on our earlier work [9], we expect an improvement in the accuracy of the results as we increase the size of the training set. Also, for this preliminary experiment, we have not considered scenarios wherein an ASFR can belong to more than one category. For instance, an ASFR such as *System should generate reports on the Complaint Register on a monthly basis through a batch process* belongs to two ASFR categories namely 'Report' and 'Batch processing'. We have excluded such ASFRs from this study. This scenario will be handled in our immediate future enhancement of ArcheR.

The categories listed in Table II are based on the analysis of our interviews with eight architects. We do not claim the exhaustiveness of these categories at this point. We expect more categories to emerge as we expand the scope of our research and design further interview studies with practitioners in other application domains.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we present a study that establishes classes of FRs that carry an implicit architectural impact but may not always be stated in a way that aids architectural decisions. Further, we present a new approach to identify and classify ASFRs from requirements documents. The results, though preliminary, are encouraging. Our immediate next step is to explore ways to extend the approach to recommend PQs and specific architectural solutions in response to the architectural impact. We are currently in the process of conducting more interviews to enrich our repository of PQs and architectural decisions. We also plan to probe into scenarios wherein a set of requirements statements together will result into a particular architectural decision. We plan to explore various machine learning techniques to detect such requirements patterns and recommend architectural solutions. We also plan to validate this approach with experts on field. We need to enrich the knowledge content of ArcheR so that we have an effective mechanism that is able to accurately flag ASFRs, indicate impact possibilities comprehensively, and offer design

solutions more pointedly. This requires a collaborative mechanism for experts to contribute and curate knowledge. We plan to harness our previous work for this purpose [23].

REFERENCES

[1] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 3nd ed., Reading, MA: Addison-Wesley, 2011.

[2] D.M. Dikel, D. Kane, and J.R. Wilson, Software Architecture: Organizational Principles and Patterns, Upper Saddle River, NJ: Prentice-Hall, 2001.

[3] C. Hofmeister, R. Nord, and D. Soni, Applied Software Architecture, Boston: Addison-Wesley, 2000.

[4] 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, pp. 1-29

[5] B. Nuseibeh, Weaving Together Requirements and Architectures. Computer, 2001. 34(3): p. 115-117

[6] 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.

[7] 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.

[8] L. Chen, M. Ali Babar, and B. Nuseibeh, Characterizing Architecturally Significant Requirements, in *IEEE Software*, vol. 30, no. 2, pp. 38–45, 2013.

[9] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, Automated classification of non-functional requirements. Requir. Eng. 12(2): 103-120 (2007)

[10] J. Cleland-Huang, IEEE Software, Thinking about Quoins: Strategic Traceability of Architecturally Significant Requirements, vol. 31, no. 4, September/October 2013, pp. 16–18.

[11] D. Ameller, X. Franch, Quark: a method to assist software architects in architectural decision-making, CIbSE 2013.

[12] Ameller D., Franch X, Ontology-based Architectural Knowledge representation: structural elements module. In: IWSSA CAiSE 2011

[13] M. A. Babar, A. W. Brown, and I. Mistrik. Agile Software Architecture: Aligning Agile Processes and Software Architectures. Morgan Kaufmann. 2013.

[14] R.N.Taylor, N.Medvidovic, E.M. Dashosy, Software Architecture, Foundations, Theory and Practice, Wiley, 2009.

[15] M. Shahin, P. Liang, M. R. Khayyambashi. A Survey of Architectural Design Decision Models and Tools. Technical Report SBU-RUG-2009-SL-01, Sheikh Bahaei University & University of Groningen, 2009

[16] http://www-03.ibm.com/software/products/en/ratidoor last accessed on 17-11-2014

[17] Bryant, K. Charmaz, The Sage handbook of grounded theory, (pp. 1-28), Sage, London, 2007.

[18] http://instructional1.calstatela.edu/lkamhis/tesl565_sp04/troy/spchact.htm Last accessed on 20-11-2014

[19] D. Ameller, C. P. Ayala, J. Cabot, X. Franch: Non-functional Requirements in Architectural Decision Making. IEEE Software 30(2): 61-67 (2013)

[20] R. Kazman, L. Bass, Toward Deriving Software Architectures from Quality Attributes, tech. report CMU/SEI-94-TR-10, Software Eng. Inst., Carnegie Mellon Univ.,1994

[21] D. Grossman and P. Domingos (2004), Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood, In Press of Proceedings of the 21st International Conference on Machine Learning, Banff, Canada

[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] S. Ghaisas, N. Ajmeri, Knowledge assisted requirements evolution, In Press, Managing requirements Knowledge, (MaRK) Springer, 2013.

[24] Wen Zhang, Ye Yang, Qing Wang, Fengdi Shu, An empirical study on classification of non-functional requirements, SEKE 2011.

[25] A. Koziolek, Architecture-driven quality requirements prioritization, in First International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks 2012). IEEE Computer Society, 2012.

[26] J.B. Lovins, Development of a stemming algorithm, Mechanical Translation and Computational Linguistics, vol.11, nos.1 and 2, March and June 1968

[27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten (2009); the WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

[28] K. Charmaz, Constructing grounded theory: A practical guide through qualitative research, Sage, Thousand Oaks, CA, 2007.

[29] Matavire, Brown, Profiling grounded theory in information systems research, European Journal of Information Systems (2013) 22, 119–129. doi:10.1057/ejis.2011.35; published online 30 August 2011

[30] J. Bosch, Design and Use of Software Architecture: Adopting and Evolving a Product-Line Approach, Boston: Addison-Wesley, 2000.

[31] 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.

[32] P. Clements, L. Northrop, Software Product Lines: Practice and Patterns, Boston: Addison-Wesley, 2002.

[33] J. Garland, R. Anthony, Large-Scale Software Architecture: A Practical Guide using UML, New York: John Wiley & Sons, Inc., 2002

[34] Gross A., Doerr J., What do Software Architects Expect from Requirements Specifications? Results of Initial Explorative Studies, IEEE First International Workshop on Twin Peaks of Requirements and Architecture, (Twin Peaks 2012), Chicago, IL