Empirical Study on the Effect of a Software Architecture Representation’s Abstraction Level on the Architecture-Level Software Understanding

Srdjan Stevanetic
Software Architecture Research Group
University of Vienna, Austria
Email: srdjan.stevanetic@univie.ac.at

Uwe Zdun
Software Architecture Research Group
University of Vienna, Austria
Email: uwe.zdun@univie.ac.atm

***Abstract*—Architectural component models represent high level designs and are frequently used as a central view of architectural descriptions of software systems. Using the architectural component model it is possible to perceive the interactions between the system’s major parts and to understand the overall system’s structure. In this paper we present a study that examines the effect of the level of abstraction of the software architecture representation on the architecture-level understandability of a software system. Three architectural representations of the same software system that differ in the level of abstraction (and hence in the number of components used in the architecture) are studied. Our results show that an architecture at the abstraction level that is sufficient to adequately maps the system’s relevant functionalities to the corresponding architectural components (i.e., each component in the architecture corresponds to one system’s relevant functionality) significantly improves the architecture-level understanding of the software system, as compared to two other architectures that have a low and a high number of elements and hence tangles or scatters the system’s relevant functionalities into several architectural components.**

I. Introduction

The software architecture of a software system is defined as “the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them” [3]. The main idea of software architecture is to concentrate on a high level view of a software system, i.e., to enable the organization of the fine-grained implementation artefacts into higher level organizational units. Architectural component and connector models (or component models for short) are frequently used as a central view of the architectural descriptions of software systems [4]. An architectural component view represents a developers’ view and it is a high-level abstraction of the entities in the source code of the software system. In the context of object-oriented designs, components group classes, as well as other components. They usually provide one or a set of similar functionalities.

Architectural understanding of a software system plays a key role in managing and maintaining the overall software system. The architectural component model of a given system can be used to perceive the interactions between the system’s major parts and to understand the overall system’s structure. So far in the software architecture literature we find only a very few studies that provide empirical evidence on the architecture-level understandability or the measurement of understandability (see e.g. [11], [8]). To the best of our knowledge, there is no existing empirical study on the understandability of architectural component models and their role in supporting the understandability of a given software system (the two previously cited studies [11], [8] examine understandability at the package level).

In this paper we present a study we carried out to examine how the architecture-level understandability of a software system is affected by the level of abstraction of the software architecture representation of the system. In particular, the participants are asked to study three architectural representations of the same software system that differ in the level of abstraction (and hence in the number of components used in the architecture) and to answer questions related to the understandability of the system. The understandability questions are constructed based on nine principal understanding activities [19] that are typically performed during real-world software understanding. The participants of our study were 56 students of the Information System Technologies lecture at the University of Vienna, Austria, in the Winter Semester 2013. Our results show that the architecture at the abstraction level that is sufficient to adequately maps the system’s relevant functionalities to the corresponding architectural components significantly improves the architecture-level understanding of the software system in comparison to the other two architectures. We also indicate the aspects that need to be further investigated in order to get a more precise insight into the studied problem.

This paper is organized as follows: In Section II, we briefly discuss the related work. In Section III we describe the study design. Section IV describes the statistical methods we applied and the analysis of our data. In Section V we discuss the threats to validity of the study. In Section VI we conclude and discuss future directions of our research.

II. Related Work

Model understandability has been studied by a number of authors in the field of data models. In that context, model understandability has been defined as the ease with which the model can be understood [17]. Moody proposes three metrics for model understandability: the model user rating of model understandability, the ability of users to interpret the model correctly, and the model developer rating of model understandability [17]. In the work by Patig [21] the variables and tasks that have been proposed by cognitive psychology or applied in computer science to test understandability are extracted. All variables have been theoretically justified by the authors that used them. In our study we measured the correctness of recovered answers.

A number of authors proposed ways to improve the understandability of architectural models through additional models or documentation artefacts. A major research direction deals with documenting architectural decisions and architectural knowledge in addition to component models [1], [13]. Another major research direction deals with architectural views [5], [12] which enable different stakeholders to view the architectures from different perspectives. Both research directions only complement component models with additional knowledge, but neither of them can fully resolve understandability issues related to the component models themselves.

Some authors emphasized the role of domain knowledge in the system understanding. Rugaber et al. [23] showed how a model on an application's domain is able to serve to programming-language-based analysis methods and tools. A domain model can provide knowledge how domain concepts are related. Domain knowledge can be captured and expressed using the so-called feature modelling. The importance of feature modelling for the system architecture is explained in the work by Pashov and Riebisch [20]. Our study further stresses the important role of features in the architecture as they represent the system's relevant functionalities that have to be captured in the architectural component models (see more details in Section III-A).

III. Empirical study description

For the study design we have followed the experimental process guidelines proposed by Kitchenham et al. [14] and Wohlin et al. [27]. The former was primarily used in the planning phase of the study while the later was used for the analysis and the interpretation of the results.

A. Goals

Despite of the precise definition of the software architecture, it is a relative concept because of the multiple levels of abstraction at which the software system can be considered [10]. From many practical examples we can see that the software architecture is created differently. Different sets of functionalities/concerns are considered to be architectural, for instance, the earliest (in time) concerns, the concerns that are more difficult (expensive) to change later on, etc [10]. All these facts emphasise the lack of the exact guidelines of how to crate appropriate architectural component models. Our study aims to provide one step toward a creating of an understandable architecture, based on the empirical evidence.

The main idea of this study is to explore how the architecture-level understandability of a software system is affected by the level of abstraction of the software architecture representation of the system. Namely, in our previous work [26] we realised that low and high numbers of elements (components and connectors) in the architectural representation of a software system decrease the architecture-level understandability of the system. In particular, we observed that the architectural understandability significantly decreases when the number of components in the architecture lies below 5 and above 15. The obtained values are considered only as very rough results, however, because they are observed in a diverse number of different systems and their component models. To reach our hypothesis, we investigated the size and functionalities of those systems in relation to their component models and concluded that the roughly predicted "optimal" range of [5,15] for component model size means that in those component models exactly one system's relevant functionality or concern is modelled by each component. As system's relevant functionalities we subsume all the objective actions and capabilities required by the user that the system must be able to perform [16]. In the text below we discuss system's features modelling which gives a little bit better insight into the system's relevant functionalities/concerns. Our previous study was based on the subjects' ratings and the qualitative explanations of their answers. In this study we aim to investigate the observed phenomenon using more objective criteria.

Modelling of the system's functionalities/concerns can be mapped to the system's features modelling in the architecture. A feature is realized functional requirement in the system, and generally also subsumes non functional requirements [7]. According to Kuusela et al. [15], two types of requirements/features are involved in the life-cycle of the software system: design objectives and design decisions, and both should be taken into account as relevant for the software architecture. The design objectives represents the functional requirements and are called design objective features while the design decisions represent the solution domain of the requirements analysis and capture the non-functional requirements. Our study aims to show that the component models where each architectural component corresponds to one system's relevant feature (so that there is no overlapping between or split of the system's relevant features in the architecture - feature scattering and tangling [25]) is preferable over more abstract or less abstract representations where the system's relevant features are overlapped or distributed over many components.

In order to adequately reflect the abstraction level of the architecture we adopted the multiplication factor 3 to create the architectures with low and high numbers of components, i.e., the architecture that adequately maps the system's functionalities to the architectural components has 9 components while the other two architectures have 3 and 28 components. [1] The architecture with 9 components is created by studying deeply the subject's system and its domain and extracting the relevant system's functionalities whereby each functionality is uniquely mapped to the corresponding architectural component. Two experienced software architects spent a couple of days in studying the system's documentation and extracting its architecture together with the traceability links that link the architectural design and its implementation. The other two architectures are created to reflect the cases of overlapping between or split of the system's relevant functionalities in the architecture (functionality scattering and tangling) which is

[1] 28 is not exactly the factor 3 from 9. The reason for that is that we found slightly more appropriate grouping of classes into 28 components in terms of their functionalities. Anyway this does not affect the study design at all.

often the case in practice as we explained above.

Our study goal has three main influencing factors: (1) the size of the architecture, i.e. the number of components in the architecture; (2) the abstraction level of the architecture (i.e. the number of components with regard to the number of system's functionalities); (3) the mapping between the system's relevant functionalities and the architectural components. Regarding the first factor, for bigger systems (the subject system used in this study can be considered as a small to medium size system) that have a lot of functionalities, the architecture that maps one-to-one the system's relevant functionalities to the corresponding architectural components would have a lot of components which might also cause understandability problems related to high cognitive load and human perception limits. In that case, it seems more suitable to use a hierarchical representation of the architecture where each level models the system's relevant functionalities at different levels of abstraction wherein the functional decomposition should reach a sufficient level of detail, i.e. provides all the system's relevant functionalities and capabilities. The explained phenomenon of hierarchical architectural decomposition is not addressed in our study and needs to be investigated further by studying bigger systems. Regarding the second factor, abstraction level of the architecture, in order to examine the architectures that have numbers of components between the values that we adopted, more studies need to be conducted. Finally, regarding the third factor, assume that we have as many components in the architecture as there are system's relevant functionalities (appropriate abstraction level) but there is a mismatch in the mapping, i.e. the components do not appropriately capture those functionalities. This phenomenon seems to have a bad effect on the understandability but needs also to be investigated further and it is not addressed in our study.

B. Variables

We differentiate 1 dependent and 5 independent variables in our study. The dependent variable in the study is the correctness of recovered answers. All the question in the study are subjective, open-ended questions. The correctness of the answers is accessed by using F-measure, the standard metric used to evaluate the performances of information retrieval systems calculated as a harmonic mean of the recall and precision measures [2]. The recall and precision measures are calculated based on the answers to the questions that consist of a list of system elements.

The independent variables used in our study concern the participants experience (programming experience, commercial programming experience, and experience in programming computer games), group affiliation (3 different groups of participants) and time spent in the study. With respect to the goal of our study 3 different treatments are defined for the participants.

The dependent variable together with its scale type, unit, and range is shown in Table I. The independent variables are shown in Table II (Please note the range for the variable "Group affiliation": "Group A3" corresponds to the participants who have studied the architecture with 3 components, "Group A9" corresponds to the participants who have studied the architecture with 9 components, and "Group A28" corresponds to the participants who have studied the architecture with 28 components).

Description	Scale type	Unit	Range
Correctness of recovered answers	Interval	Points	[0,1]

TABLE I. DEPENDENT VARIABLE

Description	Scale type	Unit	Range
Programming experience	Ordinal	Years	4 categories: 0, [1-3), [3-7), >=7
Commercial programming experience	Ordinal	Years	4 categories: 0, [1-3), [3-7), >=7
Experience in programming computer games	Ordinal	Years	4 categories: 0, [1-3), [3-7), >=7
Time	Ordinal	Minutes	90 minutes (max)
Group affiliation	Nominal	N/A	Group A3, Group A9, Group A28

TABLE II. INDEPENDENT VARIABLES

C. Hypothesis

Based on previous considerations we formulate the following hypothesis:

H_0: The architecture at the abstraction level that is sufficient to adequately map the system's relevant functionalities to the corresponding architectural components (i.e., each component in the architecture corresponds to one system's relevant functionality), significantly improves the architecture-level understanding of a software system compared to an architecture that is: 1) very abstract (hence has less elements) and tangles several system's relevant functionalities into one component or 2) very detailed (hence has more elements) and scatters system's relevant functionalities into several components.

D. Study design

The execution of the study used to test the hypothesis took place as part of the Information System Technologies lecture at the University of Vienna, Austria, in the Winter Semester 2013.

1) Subjects: The subjects of the study were 56 bachelor students of the Information System Technologies lecture at the University of Vienna.

2) Objects: The software system to be studied by participants was the Soomla Android store[2] Version 2.0, an open source framework for supporting virtual economy in mobile games. It is written in Java with which the participants were sufficiently familiar and its source code comprises of 54 source code classes distributed across 8 packages and therefore it is likely comprehensible for the participants within an study session, but not too simple.

3) Instrumentation: The following instruments were used to carry out the study:

[2]see: http://project.soom.la/

a) Architectural documentation about the Soomla Android store version 2.0: The documentation describes the conceptual architecture and lists technologies and frameworks used in the implementation. Besides text, a UML component diagram is used to illustrate the components in the system, and their inter-relationships in parts of the architecture. Participants were also provided with the set of traceability links, showing the relations between architectural components and their realized code classes.

b) Browser-based source code access: Browser-based access to the source code of Soomla Android store was provided in a Lab environment on prepared computers. All source code classes were grouped into the corresponding components so that the participants can easily study the components in the system by studying their realized source code classes.

c) A questionnaire to be filled-in by the participants during the study execution: On the first page of the questionnaire, the participants had to rate their experience, i.e. programming experience, commercial programming experience, and experience in programming computer games. The subsequent pages contain the understanding questions. In the context of the questions, two important criteria are applied: (i) the questions should be representative for key understanding and maintenance contexts, and (ii) they should be imaginatively constructed to measure the deeper understanding of the participant groups. With regard to this, nine principal understanding activities that are typically performed during real-world software understanding are applied. Please refer to [19] for the detailed description of these activities. Guided by these activities, 10 representative questions (shown in Table III) are defined that highlight many of the Soomla Android store aspects at both a high-level of abstraction (architecture-level) and a low-level of abstraction (source-code-level). The last column in the table shows the mapping between the questions and the aforementioned nine principal comprehension activities.

E. Execution

1) Preparation: As explained in Section III-D, the study was conducted at the University of Vienna, Austria in the context of a lecture on Information System Technologies. The total time limit for the whole study was 1.5 hours. The participants were randomly assigned to the three groups to ensure that the experience of the participants in all three groups is well balanced.

2) Data collection: According to the experience of the participants we can say that the participants have medium to high programming experience (most of them have [1,3) and [3,7) years of programming experience while some of them have more than 7 years of experience). Only a very few participants have industrial and game programming experience.

The data in Figure 1 reports average correctness for each study question for all three groups of participants. The figure shows that the participants of "Group A9" have a higher average correctness for all the questions except for Questions 5 and 7 than the participants of the other two groups. For Question 5 the participants of "Group A3" performed slightly better than the participants of "Group A9" while the participants of "Group A28" performed worse. The reason for this might be the fact that the participants of "Group A3" slightly better utilized the architecture than the participants of "Group A9". However, both "Group A3" and "Group A9" were able to extract the relevant information from the architectural component "GooglePlayBilling" that is present in both architectures studied by "Group A3" and "Group A9", which is not the case for "Group A28". The reason for the result in Question 7 might be that the participants in "Group A9" did not exactly know what to look for in the architecture and the corresponding traceability links, whereas the participants of the "Group A3" probably took one of the relevant classes that perform the database operations as the starting point, identified which classes are used to access the database and followed the import statement(s) in other classes to identify the answer to the question. For "Group A28" the answer to Question 7 was distributed over several components which probably hampered finding the right solution. Regarding the rest of the questions (especially Questions 3, 6, 8 and 9) inappropriate mapping between the system's relevant functionalities and the architectural components hampered the location of those functionalities in the system as well as examining the relations between them. Therefore, it hampered the overall understanding of (a subset of) a system and then directly affects answering the questions related to general software comprehension tasks [19].

ID	Description	Comprehension activities
Q1	Determining the classes that implement security and encryption/decryption functionalities	A1, A9
Q2	Determining the classes that use the services from the classes that implement security and encryption/decryption functionalities	A4, A6
Q3	Identifying the components and the corresponding classes that implement store assets and their categorization (types)	A1, A9
Q4	Identifying the components and the corresponding classes that implement pricing models for store assets	A1, A9
Q5	Investigating the impact of adding to or changing the functionality of implemented billing framework	A2, A8
Q6	Identifying the classes that communicate with the Google play service and let you sell virtual goods from your applications	A4, A6
Q7	Identifying the classes for manipulating the storage and retrieval operations in the database	A1, A9
Q8	Identifying the main class for storing/retrieving in/from the database and the strongly coupled classes to that class	A3, A4, A6
Q9	Investigating the impact of changing the database and the corresponding database services	A2, A8
Q10	Investigating common data flow during the process of billing using distributed services from the Google Play Server at runtime	A5, A7

TABLE III. QUESTIONNAIRE FOR ARCHITECTURE-LEVEL SOFTWARE UNDERSTANDING

The participants who have 0 years of programming experience are excluded from the consideration for the statistical analysis pursued in Section IV. Two of the participants (one in "Group A3" and one in "Group A28") answered just a couple of the questions and they were also excluded from the analysis because this would just introduce bias in the results.

IV. Analysis

A. Testing Hypothesis

Based on the data obtained from the questionnaire we applied the following statistical analyses:

- Testing the assumptions of parametric data: the Shapiro-Wilk normality test [24], the Levene's test for homogeneity of variance [18]
- Comparison of means between more than two variables: The one way independent ANOVA test [9]

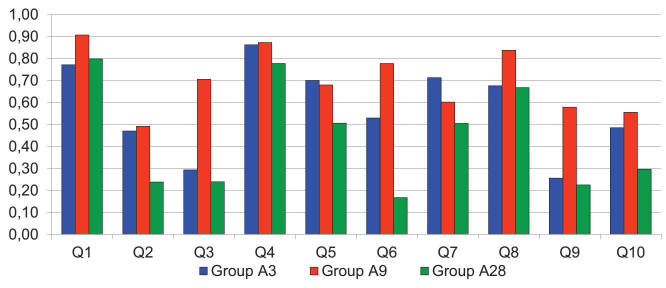

Fig. 1. Average correctness for each study question

ANOVA	Df	Sum Sq	Mean Sq	F-value	p-value
GroupID	2	60.16	30.079	17.04	**2.27e-06**
Residuals	49	88.24	1.765		

***Post hoc* test and effect size (r)**	**Diff**	**Lwr**	**Upr**	**p-value**	**r**
Group A9 - Group A3	1.2351	0.1499	2.3203	**0.0222**	0.4121
Group A28 - Group A3	-1.3495	-2.4346	-0.2642	**0.0114**	0.4585
Group A28 - Group A9	-2.5846	-3.6542	-1.5150	**0.000001**	0.7339

Df – Degrees of freedom; **Sum Sq** – Sum of squares;
Mean Sq – Mean squares; **F-value** – F ratio;
Diff - difference between means for each pair of groups;
Lwr, Upr - lower and upper limits of a 95% confidence interval for Diff

TABLE IV. ANOVA TEST, *post hoc* TEST AND EFFECT SIZE - RESULTS

For statistical analysis of the obtained data, we used the programming language R [22].

1) Testing the Assumptions of Parametric Data: In order to apply parametric tests certain assumptions must be true: data normally distributed, homogeneity of variance through the data, at least an interval level of the data, and independence of scores in the response variable(s) (i.e., what you get from one subject should be in no way influenced by what you get from any of the others) [9]. The last two assumptions are automatically fulfilled by the methodology used in the study, therefore we will focus on examining the first two assumptions.

As the first step, we tested the normality of the data by applying the Shapiro-Wilk normality test in R as well as by checking skewness, kurtosis and normal Q-Q plots for our data [9]. From the obtained results we can say that the assumption of normality of our data is not violated.

To test the homogeneity of variance the Levene's test is applied. After applying the test we obtained that the assumption of homogeneity of variance for our data is viable (F-value=0.3383, p=0.7146).

2) Comparison of Means Between More Than Two Variables: To test the hypothesis $\mathbf{H_0}$ we applied the one way independent ANOVA test. ANOVA test is a parametric test that tells us whether the means from two or more variables are the same, so the null hypothesis states that all group means are equal. The null hypothesis is tested at the significance level of 0.05. If the overall test is significant, *post hoc* tests that consists of pairwise comparisons among the three groups should be completed in order to examine which groups show significant difference against the other groups. Table IV shows the results of the ANOVA test and the corresponding results for pairwise comparisons. Also, the effect size (r) that characterizes the strength of the difference between each two groups is calculated [9].

From Table IV we see that the overall ANOVA test is significant (p-value=2.27e-06). *Post hoc* test shows that there is a significant difference between "Group A3" and "Group A9" as well as between "Group A28" and "Group A9" (p-values<0.05). Furthermore there is a significant difference between "Group A3" and "Group A28" (see Table IV) which suggests that the architecture with 3 components still provides useful information about the system's structure in comparison to the architecture with 28 components that distributes the system's relevant functionalities across many components and represents very partitioned design. Regarding the values for the effect size the differences between "Group A3" and "Group A9" and between "Group A3" and "Group A28" are medium while the difference between "Group A9" and "Group A28" is large [9].

Given the results from the analysis undertaken, it has been demonstrated that the hypothesis $\mathbf{H_0}$ of our study is supported.

V. Validity evaluation

In this section we discuss the various threats to validity of our study and how we tried to minimize them:

a) Conclusion validity: The conclusion validity defines the extent to which the conclusion is statistically valid. The statistical validity might be affected by the size of the sample (17, 18, and 17 students in the groups). In a between subjects-design, 20 participants are recommended to detect a large effect in the one way ANOVA test with a power of 0.8 and a significance level of 0.05 [6]. As we obtained that there is a statistically significant difference between the studied groups (with a medium and a large effect size) for the given sample size we would be able to detect even tiny differences between the groups if the sample size increases. Therefore there is a low threat to conclusion validity of our results.

b) Internal validity: The internal validity is the degree to which conclusions can be drawn about cause-effect of independent variables on the dependent variables.

A potential threat to validity might be that the understanding of the questionnaire could have been biased towards "Group A9". Answering some of the questions might be easier for the "Group A9" because the architecture for that group reduces the decision space by pointing to the component or the set of components that implement the examined functionality. However, those questions represent a main part of the established comprehension framework related to examining the relevant functionality of (a part of) the system and how the identified functionalities are interrelated [19]. The established task framework also ensures that many aspects of typical understanding contexts are covered. As a result, the questionnaire concerned both global and detailed knowledge, as well as static and dynamic aspects. Therefore, we do not consider it a highly relevant threat to validity.

c) External validity: The external validity is the degree to which the results of the study can be generalized to the broader population under study.

The participants' population in the study might not be sufficiently competent. This might influence the results of the study. In this study, all the participants had knowledge about software development and software architecture (UML modelling), as well as of software traceability. They all studied the previous lectures of at least the software architecture course and have medium to high programming experience. However we are aware that more empirical studies with professionals need to be carried out in order to generalize the results.

VI. Conclusions and Future Work

In this paper we present the empirical study that examines how the architecture-level understandability of a software system is affected by the level of abstraction of the software architecture representation of the system. The subjects of the study were 56 students of the Information System Technologies lecture at the University of Vienna, Austria. They were divided into three groups each of them studied one of three architectural representations of the same system that differ in the level of abstraction (and hence number of components in the architecture). Our results show that the architecture at the abstraction level that is sufficient to adequately map the system's relevant functionalities to the corresponding architectural components (i.e., each component in the architecture corresponds to one system's relevant functionality) significantly improves the architecture-level understanding of the software system, as compared to two other architectures that have a low and a high number of elements and hence a very abstract or very detailed mapping to system's relevant functionalities (the scaling factor 3 is (roughly) used to create the architectures with lower and higher numbers of elements). In other words it means that tangling several system's relevant functionalities into one component or scattering them into several architectural components significantly decrease architectural understandability. Improving our understanding of how to model architecture has a great value and helps to improve the quality of the software it represents.

Acknowledgement

This work was supported by the Austrian Science Fund (FWF), Project: P24345-N23. We thank Dr. Nina Senitschnig from the Department of Statistics and Operations Research, University of Vienna, Austria, for valuable suggestions and help related to the statistical analysis pursued in the study.

References

[1] M. A. Babar and P. Lago. Editorial: Design decisions and design rationale in software architecture. *J. Syst. Softw.*, 82(8):1195–1197, Aug. 2009.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[4] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Boston, MA, 2003.

[5] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.

[6] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. L. Erlbaum Associates, 1988.

[7] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Trans. Softw. Eng.*, 29(3):210–224, Mar. 2003.

[8] M. O. Elish. Exploring the relationships between design metrics and package understandability: A case study. In *ICPC*, pages 144–147. IEEE Computer Society, 2010.

[9] A. Field. *Discovering Statistics Using SPSS*. SAGE Publications, 2005.

[10] B. Graaf. Model-driven evolution of software architectures. In *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*, pages 357–360, March 2007.

[11] V. Gupta and J. K. Chhabra. Package coupling measurement in object-oriented software. *J. Comput. Sci. Technol.*, 24(2):273–283, Mar. 2009.

[12] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley Professional, 2000.

[13] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, pages 109–120, Washington, DC, USA, 2005. IEEE Computer Society.

[14] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, 28(8):721–734, Aug. 2002.

[15] J. Kuusela and J. Savolainen. Requirements engineering for product families. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 61–69, 2000.

[16] C. Mazza, J. Fairclough, M. Bryan, P. Daniel, S. Adriaan, S. Richard, J. Michael, and G. Alvisi. *Software Engineering Guides*. Prentice-Hall International (UK), 1996.

[17] D. L. Moody. Metrics for evaluating the quality of entity relationship models. In *Proceedings of the 17th International Conference on Conceptual Modeling*, ER '98, pages 211–225, London, UK, UK, 1998. Springer-Verlag.

[18] I. Olkin. *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*. Stanford studies in mathematics and statistics. Stanford University Press, 1960.

[19] M. J. Pacione, M. Roper, and M. Wood. A novel software visualisation model to support software comprehension. In *Proceedings of the 11th Working Conference on Reverse Engineering*, WCRE '04, pages 70–79, Washington, DC, USA, 2004. IEEE Computer Society.

[20] I. Pashov and M. Riebisch. Using feature modeling for program comprehension and software architecture recovery. In *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, pages 406–417, May 2004.

[21] S. Patig. A practical guide to testing the understandability of notations. In *Proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling - Volume 79*, APCCM '08, pages 49–58, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[22] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2013.

[23] S. Rugaber. The use of domain knowledge in program understanding. *Ann. Softw. Eng.*, 9(1-4):143–192, Jan. 2000.

[24] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 3(52), 1965.

[25] P. Sochos, M. Riebisch, and I. Philippow. The feature-architecture mapping (farm) method for feature-oriented development of software product lines. In *Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on*, pages 9 pp.–318, March 2006.

[26] S. Stevanetic, M. A. Javed, and U. Zdun. Empirical evaluation of the understandability of architectural component diagrams. In *Companion Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, WICSA 2014, Sydney, Australia, 2014. IEEE Computer Society.

[27] C. Wohlin. *Experimentation in Software Engineering: An Introduction*. The Kluwer International Series in Software Engineering. Kluwer Academic, 2000.