Exploring the Relationships between the Understandability of Components in Architectural Component Models and Component Level Metrics

Srdjan Stevanetic and Uwe Zdun
Software Architecture Research Group
University of Vienna, Austria
srdjan.stevanetic|uwe.zdun@univie.ac.at

ABSTRACT

Architectural component models represent high level designs and are frequently used as a central view of architectural descriptions of software systems. The components in those models represent important high level organization units that group other components and classes in object-oriented design views. Hence, understandability of components and their interactions plays a key role in supporting the architectural understanding of a software system. In this paper we present a study we carried out to examine the relationships between the effort required to understand a component, measured through the time that participants spent on studying a component, and component level metrics that describe component's size, complexity and coupling in terms of the number of classes in a component and the classes' relationships. The participants were 49 master students, and they had to fully understand the components' functionalities in order to answer 4 true/false questions for each of the 7 components in the architecture of the Soomla Android store system. Correlation, collinearity and multivariate regression analysis were performed. The results of the analysis show a statistically significant correlation between three of the metrics, number of classes, number of incoming dependencies, and number of internal dependencies, on one side, and the effort required to understand a component, on the other side. In a multivariate regression analysis we obtained 3 reasonably well-fitting models that can be used to estimate the effort required to understand a component. In our future work we plan to study more components and investigate more metrics and their relationships to the understandability of components and architectural component models.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures; D.2.8 [**Software Engineering**]: Metrics

EASE '14, May 13 - 14 2014, London, England, BC, United Kingdom

General Terms

Experimentation, Measurement, Design

Keywords

Architectural Component Models, Understandability, Software Metrics, Empirical Evaluation

1. INTRODUCTION

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, especially in the case of large-scale software systems. The software architecture of the 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].

Architectural component and connector models (or component models for short) are frequently used as a central view of the architectural descriptions of software systems [6]. The main idea of a component-based software development is to manage the increasing complexity of software systems as well as to enable the reuse of software components, and in that way to facilitate the process of software development. In the context of object-oriented designs, components represent important high level organization units that group classes, as well as other components, and provide one or a couple of similar system functionalities.

Architectural understanding of a software system plays a key role in managing and maintaining the overall software system. Hence, understanding of components and their interactions in component models plays a key role in supporting the architectural understanding of a software system. So far in the software architecture literature we find only a very few studies that provide empirical evidence regarding the architectural 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 (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 the relationships between the effort required to understand a component measured through the time that participants spent on studying a component and component level metrics that describe the component's size, complexity and coupling

in terms of the number of classes in a component and the classes' relationships. The execution of the study took place as part of the Advanced Software Engineering (ASE) master lecture at the University of Vienna in the Winter Semester 2013. The subjects of the study were the 49 master students of the ASE lecture. The software system to be studied by participants was the Soomla Android store Version 2.0, an open source framework for supporting virtual economy in mobile games. In order to answer 4 true/false questions for each component the participants had to fully understand the functionalities of each component by exploring the relationships (together with their roles) between the classes within each component and the relationships that those classes have with the classes outside of the given component.

The results of our analysis show a statistically significant correlation between 3 of the metrics, number of classes, number of incoming dependencies, and number of internal dependencies, on one side, and the effort required to understand a component, on the other side. In a multivariate regression analysis we obtained 3 reasonably well-fitting models that can be used to estimate the effort required to understand a component.

This paper is organized as follows: In Section 2, we briefly discuss the related work. In Section 3 we describe the study design. Section 4 describes the statistical methods we applied and the analysis of our data. In Section 5 we discuss the threats to validity of the study. In Section 6 we conclude and discuss future directions of our research.

2. RELATED WORK

As we mentioned before, so far in the software architecture literature we find only a very few studies that provide empirical evidence regarding architectural understandability. In particular, one existing study examines the influence of package coupling on the understandability of the software systems [11], while another one examines the relationships between some package-level metrics and package understandability [8]. None of these studies focuses on understandability of architectural components, the main focus of our study.

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 [24]. 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 [24]. In the work by Patig [25] 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 the answers and the time that participants spent on resolving the questions. Furthermore, our study aims to examine the understandability of components functionalities in the system implementation which can help in designing the component models together with their corresponding mappings to the implementation with a sufficient level of understandability.

Even though so far there are no rigorous empirical studies of architectural component model understandability, aspects like fault density and reuse of components have been studied before. Fenton and Ohlsson have studied the relations of fault density and component size [10]. Mohagheghi et al. have studied the comparison between software reuse and defect density and stability [23]. Their study is based on the historical data on defects, modification rate, and software size. Malaiya and Denton identify the component partitioning and implementation as influencing factors to determine the "optimal" component size with regard to fault density [21].

Many different software metrics for measuring the system's architecture, components as its constituting parts, and structures similar to architectural component models, such as other higher-level software structures (package, module, graph-based structures) have been proposed. Metrics related to components and the corresponding architectures [15, 31, 29, 28] measure size, coupling, cohesion, and dependencies of individual components but also the complexity of the whole architecture when all the components and their interactions are taken into account. Different authors have proposed different package level metrics that measure their size, coupling, stability, and cohesion [8, 11, 22, 12, 32]. Module level metrics [27, 18, 14] measure coupling, cohesion, and size of modules, their hierarchical structure, the quality of modularization in the system, etc. Graph-based metrics measure different interactions between the nodes in the graph [13, 5, 19, 20, 2]. Some of the graph-based metrics have been shown to be useful in measuring large scale software systems in the sense that those systems share some properties that are common for complex networks across many fields of science [19]. All the mentioned metrics can be applied or can be more-less easily adapted to be applicable for the component models. However, none of the metrics is empirical validated regarding understandability of architectural components or architectural component models so far.

3. EMPIRICAL STUDY DESCRIPTION

For the study design we have followed the experimental process guidelines proposed by Kitchenham et al. [16] and Wohlin et al. [30]. 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.

3.1 Goals, variables, and hypotheses

The main idea of this study is to explore the relationships between the understandability of the components in architectural component models and component level metrics that can be used to characterize their size, complexity and coupling to the other components in the system. The metrics that we used in this study are:

- **Number of Classes (NC)**: The NC metric for a component is defined as a total number of classes inside a component.
- **Number of Incoming Dependencies (NID)**: The NID metric for a component is defined as a total number of dependencies between the classes outside of a component and the classes inside a component that are used by those outside classes.
- **Number of Outgoing Dependencies (NOD)**: The NOD metric for a component is defined as a total number of dependencies between the classes inside a com-

ponent and the classes outside of a component that are used by those inside classes.

- **Number of Internal Dependencies (NIntD)**: The NIntD metric for a component is defined as a total number of dependencies between the classes within a component.

The first three metrics are adapted from the corresponding package level metrics (number of classes for a package, package afferent coupling and package efferent coupling) defined by Martin [22]. We consider the dependencies between the components in terms of the dependencies between the classes while in the work by Martin the dependencies between packages are considered through the number of packages that are related to the given package. The first three metrics characterize the coupling and the size of a component and the fourth metric is introduced to model the internal complexity of the component in terms of the number of dependencies between classes within a component.

We can differentiate two groups of variables in our study. The first group of variables was collected from the participants of the study while the second group of variables was collected from the studied system. The first group of variables includes three independent variables related to demographic information: programming experience, commercial programming experience, and experience in programming computer games. It also includes the two dependent variables, time required to study a component and the percentage of the correct answers on the study questions. The time variable is measured by the time that the participants spent on studying each component, and it is used to measure the effort required to understand a component. The percentage of the correct answers is introduced to help in estimating the understandability effort in case that the participants do not spend enough time to deeply study all the components in the system in order to achieve a high percentage of the correct answers. Namely, if the participants do not spend enough time on studying the given component, the percentage of the correct answers will probably decrease for that component so there is a dependency relation between these two variables. Therefore, the percentage of the correct answers can assist in estimating the time required to fully study the given component (i.e., to achieve 100 % of the correct answers), which can be used as an indicator of the effort required to fully understand a component [1]. The second group of variables include the variables related to the metrics that we aim to explore: number of classes (NC), number of incoming dependencies (NID), number of outgoing dependencies (NOD), and number of internal dependencies (NIntD). They are all independent variables.

Regarding the NC metric, we expect that the bigger the size of a component in terms of the number of classes the more effort is required to understand it and therefore the more time is needed to study it. Regarding the NID metric, we expect that the higher the incoming dependencies of a component the more effort is required to understand it. This expectation can be explained as follows: high NID values for a component indicate that it is a service component that offers many services and therefore more effort is required to understand all its services. Regarding NOD metric, we expect that higher the outgoing dependencies of a component the more effort is required to understand it because it has a lot of dependencies to the outside classes that need to be understood, too. Finally, we expect for the NIntD metic that the higher the number of dependencies between the classes within a component is, the more effort is required to understand it.

The dependent variables together with their scale types, units, and ranges are shown in Figure 1. The independent variables are shown in Figure 2.

Description	Scale type	Unit	Range
Time	Ratio	Minutes	Positive natural numbers including 0
Percentage of the correct answers	Ratio	-	[0,100] 0 - the lowest, 100 - the highest

Figure 1: Dependent variables

Description	Scale type	Unit	Range
Programming experience	Ordinal	Years	4 categories: [0,1),[1-3),[3-7), >=7
Commercial programming experience	Ordinal	Years	4 categories: [0,1),[1-3),[3-7), >=7
Experience in programming computer games	Ordinal	Years	4 categories: [0,1),[1-3),[3-7), >=7
Number of Classes (NC)	Ratio	Class	Positive natural numbers including 0
Number of Incoming Dependencies (NID)	Ratio	Dependency	Positive natural numbers including 0
Number of Outgoing Dependencies (NOD)	Ratio	Dependency	Positive natural numbers including 0
Number of Internal Dependencies (NIntD)	Ratio	Dependency	Positive natural numbers including 0

Figure 2: Independent variables

Based on previous considerations we formulate the following set of hypotheses:

H_{01}: There is a significant positive correlation between the number of classes (NC) in a component and the effort required to understand a component measured though the time spent on studying it.

H_{02}: There is a significant positive correlation between the number of incoming dependencies (NID) of a component and the effort required to understand a component measured though the time spent on studying it.

H_{03}: There is a significant positive correlation between the number of outgoing dependencies (NOD) of a component and the effort required to understand a component measured though the time spent on studying it.

H_{04}: There is a significant positive correlation between the number of internal dependencies (NIntD) of a component and the effort required to understand a component measured though the time spent on studying it.

3.2 Study design

The execution of the study used to test the hypothesis took place as part of the Advanced Software Engineering (ASE) master lecture at the University of Vienna in the Winter Semester 2013.

3.2.1 Subjects

The subjects of the study are the 49 master students of the ASE lecture.

[1]Predicting the percentage of the correct answers variable is also possible but since our focus is on estimating the time variable that is used as an indicator of the effort required to understand a component we consider the percentage of the correct answers as an auxiliary variable for the time prediction as it is explained in the context.

3.2.2 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 allows mobile game developers to easier implement virtual currencies (tokens, coins, gems, etc.), virtual goods, and in-app purchases. The choice of using this particular system is motivated by the following factors:

- The Soomla Android store is a free open source system, which enables us to conduct the study and disseminate its results.
- It is used in real-world games and therefore it has industrial relevance.
- It addresses a well-known application domain that is likely known to the subjects from familiar real-world game applications.
- It is written in the Java programming language with which the participants were sufficiently familiar.
- The source code of the Soomla Android store adheres to coding standards and is rather easy to understand for most potential subjects. The source code classes are well designed in terms that there are no big deviations in their sizes, i.e. each of them provides one or a couple of similar functionalities.
- The overall source code of the Soomla Android store comprises of 54 source code classes distributed across 8 packages, containing a total of 3623 KLOC (excluding blank lines and commented lines); that is, it is likely comprehensible for the participants within an study session, but not too simple.
- The experimenters were familiar with the internals of the Soomla Android store.

3.2.3 Instrumentation

The following instruments were used to carry out the study:

Three pages of 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 interrelationships 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.

The architectural design of the Soomla Android store system in the form of UML component diagram is shown in Figure 3. It comprises of seven components, namely *Security (C1)*, *CryptDecrypt (C2)*, *PriceModel (C3)*, *GooglePlayBilling (C4)*, *StoreController (C5)*, *DatabaseServices (C6)*, and *StoreAssets (C7)*. In addition, there are two external components modelled: *GooglePlayServer*, the REST Web Services running at Google, and *SQLLiteDatabase*, the used database accessed over JDBC. A short description of the components' roles in the system is shown in Figure 4.

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

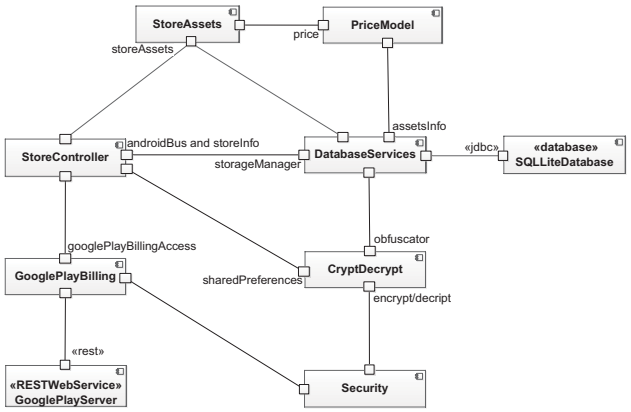

Figure 3: UML component diagram of the Soomla Android store system

Component	Component's role
Security (C1)	Verifies the information during the purchasing process
CryptDecrypt (C2)	Provides encrypt/decrypt services to obfuscate the billing information and to encrypt/decrypt the data stored to or retrieved from the database
PriceModel (C3)	Describes the model that explains how the prices of virtual items are formed
GooglePlayBilling (C4)	Simplifies in-app billing API which is a Google play service that lets you sell virtual goods from inside your applications
StoreController (C5)	Provides the runtime functionality of the Android store and contains up-to-date store information
DatabaseServices (C6)	Performs the initialization of the database and implement retrieve, add, and remove operations for store assets in the database
StoreAssets (C7)	Describes the virtual items used in the application (virtual currency, virtual goods, and their classification)

Figure 4: Soomla Android store components and their roles in the system

Browser-based source code access.

The 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 in the corresponding components so that the participants can easily study the components in the system by studying their realized source code classes.

A questionnaire to be filled-in by the participants during the experiment.

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 order to answer the questions correctly the participants had to fully understand the functionalities of each component by exploring the relationships (together with their roles) between the classes within each component and the relationships that those classes have with the classes outside of the given component. There were 4 true/false questions for each component, and the participants had to check the right answers among them. In the case of bigger components, answering the questions requires studying of more classes than in the case of smaller components. We also provided a table where the participants had to enter the time slots during which they studied each of the components. Each time slot contains a start and stop time, indicating the time when the participants started studying the given component and the time when they finish it, respectively. There were more time slots in case the participants wanted to study the component more than one

time. The time is written in the format *hour : minute*.

3.3 Execution

3.3.1 Preparation

As it is explained in Section 3.2 the study was conducted at the University of Vienna, Austria in the context of a lecture on Advanced Software Engineering. The total time limit for the whole study was 1.5 hours.

3.3.2 Data collection

The data collection was performed as planned in the design. The relevant data regarding the participants' demographic information are shown in Figure 5.

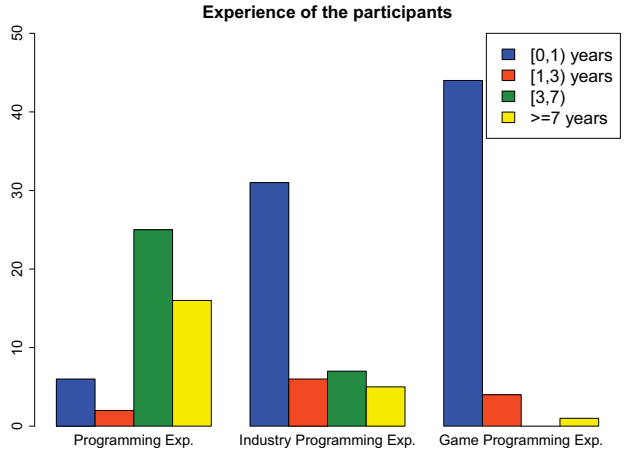

Figure 5: Experience of the participants

According to the experience of the participants we can say that the participants have medium to high programming experience (most of them have [3,7) and more than 7 years of programming experience). Many of them have industrial programming experience, while only a very few have experience in game programming.

	Security (C1)	Crypt Decrypt (C2)	Price Model (C3)	GooglePlay Billing (C4)	Store Controller (C5)	Database Services (C6)	Store Assets (C7)
Number of excl. participants	9	8	8	7	23	18	14

Figure 6: Number of excluded participants for each component

The mean, the median and the standard deviation of the time and the percentage of the correct answers variables collected from the participants are shown in Figures 7 and 8. The participants with [0,1) years of programming experience were excluded from the consideration. Some participants did not write the time they spent on studying the particular components (they did not write the start time or the stop time or both of them) and those participants were excluded from the consideration for those particular components. Some of them spent very short time in studying components which is not enough [3] and can just introduce bias in the results and those participants were also excluded from the consideration for the given component. The number of excluded participants for each component is shown in Figure 6. From the rest of the results regarding the time variable (see Figure 7) we can say that they quite well reflect our expectations[4].

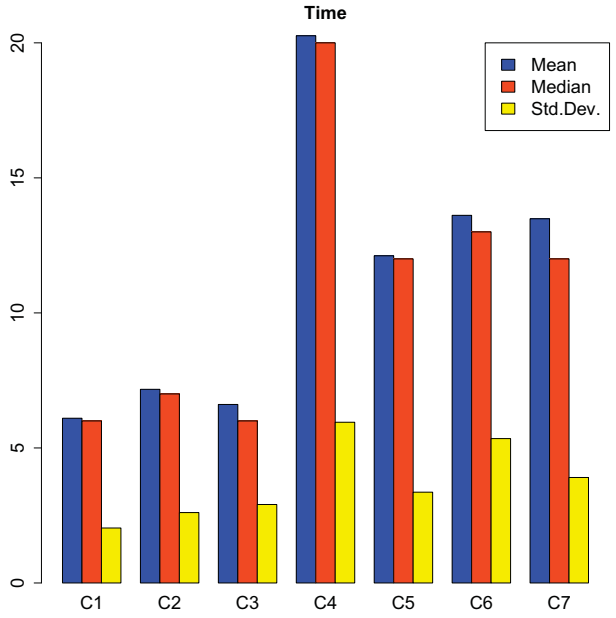

Figure 7: Time – descriptive statistics

The data related to the component level metrics are shown in Figure 9.

By looking at Figure 7 we can observe that the time needed to study the first three components is significantly decreased comparing to the other components in the system. This is an expected result because those 3 components are smaller in terms of the number of classes they contain. It can be also observed that the time the participants spent on studying the components *C5*, *C6* and *C7* is decreased compared to the component *C4*. The percentage of the correct answers (see Figure 8) for the components *C5*, *C6* and *C7* is also decreased compared to the component *C4* and the smaller components *C1*, *C2* and *C3*. Even though it is realistic to expect that the percentage of the correct answers for the bigger components decrease on average (because of the higher amount of information that need to be studied which increases the probability of missing some information), it seems also that the participants needed a bit more time for studying the components *C5*, *C6* and *C7* (or at least for studying the component *C7* which has more classes than the component *C4*) in order to achieve the higher percentage of the correct answers. Based on this consideration we obtained some prediction models for the time variable that use both the component level metrics and the percentage of the correct answers. Using that models it is possible to

[3]We expect that at least 30 seconds is needed to study each class in the component.

[4]We tried the same study with a couple of our colleagues before we tried it within the course in order to estimate how much time the participants approximately need to study the components and to ensure that there will be enough time to study all the components within the study session of 1.5 hours.

predict the time variable in order to achieve the maximum percentage of the correct answers which represents the effort required to fully understand a component. Model prediction analysis is explained in Section 4.3.

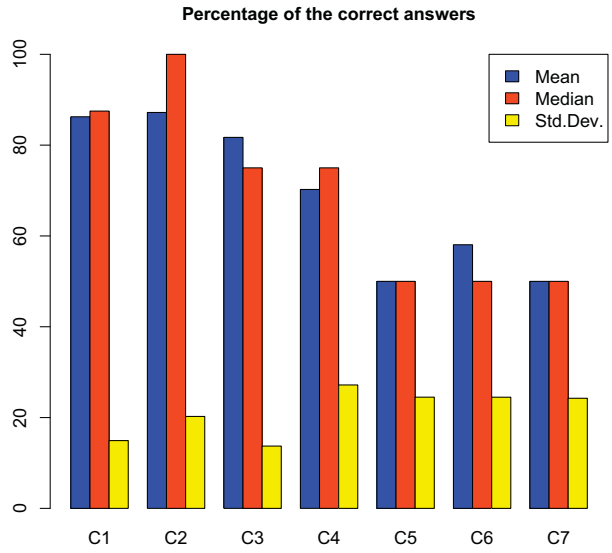

Figure 8: Percentage of the correct answers – descriptive statistics

	Number of Classes	Number of Incoming Dependencies	Number of Outgoing Dependencies	Number of Internal Dependencies
Security (C1)	2	3	4	1
CryptDecrypt (C2)	5	9	0	5
PriceModel (C3)	3	1	4	2
GooglePlayBilling (C4)	11	4	3	12
StoreController (C5)	8	5	15	5
DatabaseServices (C6)	8	8	8	13
StoreAssets (C7)	13	9	3	14

Figure 9: Component level metrics

3.3.3 *Validation*

At least one observer was present in the room during the whole study execution time to enable the participants to pose clarification questions and assure that participants did not use forbidden materials and did not talk to each other. After execution, all materials were collected before any of the participants left the room. There were no situations in which participants behaved unexpectedly.

4. ANALYSIS

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

- Normality analysis: The Shapiro-Wilk normality test
- Correlation analysis: The Spearman rank correlation test
- Collinearity analysis: The Variance Inflation Factor (VIF) and the Condition Number (CN) calculation
- Multivariate Regression analysis: Construction of multivariate linear regression models that can predict the effort required to understand a component

For statistical analysis of the obtained data, we used the programming language R [26].

4.1 Testing Hypotheses

4.1.1 *Testing the Normality*

As the first step in analysing the data, we test the normality of the data by applying the Shapiro-Wilk normality test in R. The null hypothesis H_0 for the test states that the input data are normally distributed. H_0 is tested at the significance level of $\alpha = 0.05$ (i.e., the level of confidence is 95%).

After applying the normality tests we found that all variables do not fit the normal distribution (all p-values are less than 0.05; that is the null hypothesis can be rejected). Based on that we decided to pursue the non-parametric Spearman rank correlation test with our data in the next step of the analysis. The results of the Shapiro-Wilk test are shown in Figure 10.

	Shapiro-Wilk Normality Test
Time	W = 0.9118, **p-value = 3.877e-11**
Percentage of the correct answers	W = 0.8651, **p-value = 3.273e-14**
Number of Classes	W = 0.8847, **p-value = 4.971e-13**
Number of Incoming Dependencies	W = 0.8451, **p-value = 2.667e-15**
Number of Outgoing Dependencies	W = 0.7617, **p-value < 2.2e-16**
Number of Internal Dependencies	W = 0.8164, **p-value < 2.2e-16**

Figure 10: Shapiro-Wilk Normality Test

4.1.2 *Testing the Correlation Between the Variables*

In order to test our hypotheses the Spearman rank correlation test is used with a level of significance $\alpha = 0.05$. It examines whether there is a linear correlation between the tested variables.

	Time
Number of Classes	r=0.7350 **p-value<2.2e-16**
Number of Incoming Dependencies	r=0.2575 **p-value= 3.0e-05**
Number of Outgoing Dependencies	r= -0.007 p-value= 0.9058
Number of Internal Dependencies	r= 0.6591 **p-value<2.2e-16**

Figure 11: The Spearman correlation coefficients and corresponding p-values between the time variable and the component level metrics

Figure 11 shows the Spearman's coefficients and the corresponding p-values between the time that the participants spent on studying the components and the component level metrics. There is a significant positive correlation between the variables number of classes, number of incoming dependencies and number of internal dependencies on one side and the time variable on the other side. It means that we can accept the hypotheses $\mathbf{H_{01}}$, $\mathbf{H_{02}}$, and $\mathbf{H_{04}}$ of our study, i.e. there is a significant positive correlation between the variables number of classes (NC), number of incoming dependencies of a component (NID), and number of internal dependencies of a component (NIntD) on one side and the effort required to understand a component measured though the time spent on studying it on the other side.

Regarding the hypothesis $\mathbf{H_{03}}$, we found that there is no significant correlation between the number of outgoing dependencies and the time variable. Hence, we can reject the hypothesis $\mathbf{H_{03}}$, i.e. there is no significant positive correlation between the number of outgoing dependencies (NOD) of a component and the effort required to understand a component measured though the time spent on studying it. The

correlation coefficient even becomes negative. It is a bit surprising result but can be however explained: high NOD of a component indicates high reusability of services provided by other components. If those services are well-understood they will decrease the time needed to understand the reusing classes in other components.

These results coincide with the results obtained in the work by Elish [8]. He studied package level metrics and found a significant positive correlation between the package level metrics number of classes in the package and package afferent coupling (incoming dependencies) with the effort required to understand a package. Also a non-significant negative correlation between the metric package efferent coupling (outgoing dependencies) and the time required to understand a package is found. Our study differs from the work by Elish in the following: firstly, our focus is on studying the understandability of architectural component models while he studied the understandability at the package level[5]. Secondly, we measure the understandability effort using the time that participants need to answer the questions related to the understandability of the components' functionalities while in the work by Elish the participants rated the effort they needed to understand the packages. Finally, we used the component level metrics that describe their size, coupling and complexity in terms of the number of classes they have and the classes' relationships while in the work by Elish package level metrics are used.

4.2 Collinearity Analysis

To create prediction models that can be used to predict the time variable, first we have to conduct a collinearity analysis between the variables that can be the possible predictors of the time variable and to exclude those variables that are highly correlated with other possible predictors. All possible predictors include the component level metrics and the percentage of the correct answers (according to the discussion in Section 3.3.2). Accordingly, the Condition Number (CN) and the Variance Inflation Factor (VIF) values for the metrics were calculated. If the VIF values are higher than 10, multicollinearity is strongly suggested. The acceptable values for the condition number are the values less than 30 (a threshold suggested in the literature [4]). Figure 12 shows the CN and VIF analysis results.

Variable	VIF	VIF (without NIntD)
Percentage of the correct answers	1.4405	1.4391
Number of Classes (NC)	7.3977	1.6092
Number of Incoming Dependencies (NID)	1.5776	1.4213
Number of Outgoing Dependencies (NOD)	1.2432	1.2135
Number of Internal Dependencies (NintD)	7.9627	N/A
Condition number (CN)	5.72	4.94

Figure 12: Condition Number and Variance Inflation Factor analysis results

As we can see from Figure 12, the VIF results for the case of all predictors are less than 10 and the greatest VIF value is 7.96 (for NIntD). It seems that there is a tendency of multicollinearity between the variables (not so strong). Predictor that tends to be highly correlated with some other predictors (in our case this is NIntD, which has the greatest VIF value) can be linearly predicted from the others and it is redundant in the model. Therefore we decided to exclude the NIntD from our model after which we get acceptable results for both VIF and CN values (see Figure 12).

[5]Architectural component models are more graph based structures where the nodes are the components that include realized classes and the edges reflect the communication between components while package models have more hierarchical-based structure (sub-packages at different hierarchical levels).

4.3 Multivariate Regression Analysis

Multivariate regression analysis is performed to construct different multivariate regression models that can be used to predict the effort required to understand a component measured through the time that participants spent on studying a component. We used the Mallows' Cp calculation to create reasonably fitting models that prevent over-fitting of the data [17]. All the models that have $Cp \leq p$ (p - number of predictors including the constant predictor, if present) must be considered reasonably good fits. In Figure 13, the Mallows' Cp parameter versus the number of predictors including the constant predictor (if present) for models with different combinations of the possible predictors (all variables in Figure 12 except NIntD) are shown. The drawn curve in the figure is the curve $Cp = p$. We can see that 3 models fit the explained criteria ($Cp \leq p$), i.e. they lie below or on the line $Cp = p$.

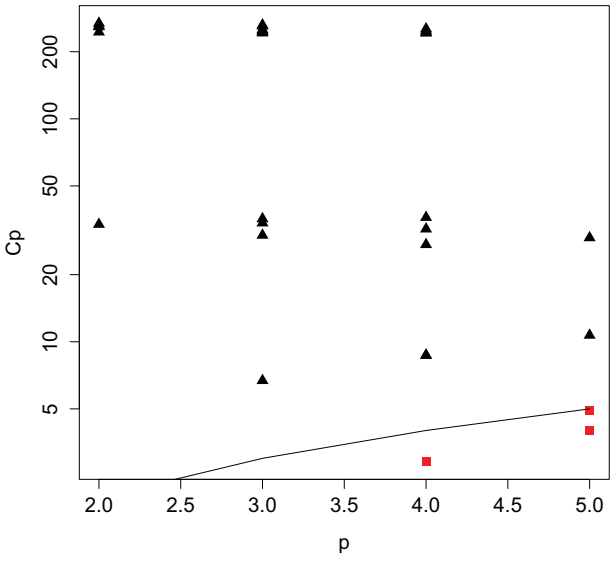

Figure 13: Mallows' Cp results

The accuracy of the predicted models is checked using different results in R. The residuals are checked to follow the normal distribution. The influential points are the points whose removal will cause a large change in the fit [9]. Those points can be detected using the Cook's distance contour lines. If some points have a distance larger than 1, it suggests the presence of a possible outlier or a poor model. The obtained models do not have such influential points. The coefficient of determination (R^2) is used to describe how well the regression fits a set of data. The statistical test of significance for R^2 is the F test [7]. The Mean Magnitude of Relative Error (MMRE) and prediction at level 0.25 (Pred(0.25)) measures are calculated as de facto stan-

dard and commonly used measures for the evaluation of the accuracy of the predicted models. The models coefficients, adjusted R^2, p-value for the F test, MMRE, and Pred(0.25) values are shown in Figure 14.

	Adjusted R-squared	F-statistic: p-value	MMRE	Pred(0.25)
Model 1	0.8801	< 2.2e-16	0.3522	0.4687
Model 2	0.8811	< 2.2e-16	0.3527	0.4921
Model 3	0.8813	< 2.2e-16	0.3579	0.5117

Coefficients	Intercept	Percentage of the correct answers	Number of Classes	Number of Incoming Dependencies	Number of Outgoing Dependencies
Model 1	x	4.8597	1.5162	-0.5349	x
Model 2	x	4.5754	1.4628	-0.5175	0.1150
Model 3	2.4250	2.8902	1.4200	-0.5795	x

Figure 14: Models' parameters

For all the models adjusted R^2 is around 88 %, MMRE is around 35 %, and Pred(0.25) is in the range [46,51] %. Those results suggest that the obtained models fit the data quite well. The best model in terms of Pred(0.25) is Model 3, which has a value of 51 %. Beside the Mallows' Cp calculation we also tested our models using the 10-fold cross-validation technique which is also useful for overcoming the problem of over-fitting [1]. The results show that the 3 previously selected models perform the best among the other models and have almost the same MMRE and Pred(0.25) values as it is shown in Figure 14. It confirms the validity and the results of our previous analysis using the Mallows' Cp criterion.

Using the obtained prediction models we can calculate the time variable in order to achieve the maximum percentage of the correct answers (100 %) which represents the effort required to fully understand a component. In Figure 15 the predicted time variable using the model with the least number of predictors and the time variable obtained from the participants are shown. The predicted time variable slightly differs from the time variable obtained from the participants. Just for the component *StoreAssets (C7)* the difference is significant. It can be interpreted in the way that the participants needed a bit more time for studying the component *StoreAssets (C7)* while they spent enough time for studying the other 6 components in order to be able to answer all the questions correctly.

In order to obtain more robust prediction models that can be used in a broader sense we need to study more systems and to explore other software metrics that can assist in assessing the understandability of components and architectural component models.

5. VALIDITY EVALUATION

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

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 (49 students answered the questions for the 7 components).

The maximum number of participants we excluded from the consideration for some component is 23 (see Figure 6). This means that the minimum number of participants who are considered for any component is 26 which is quite fair. In

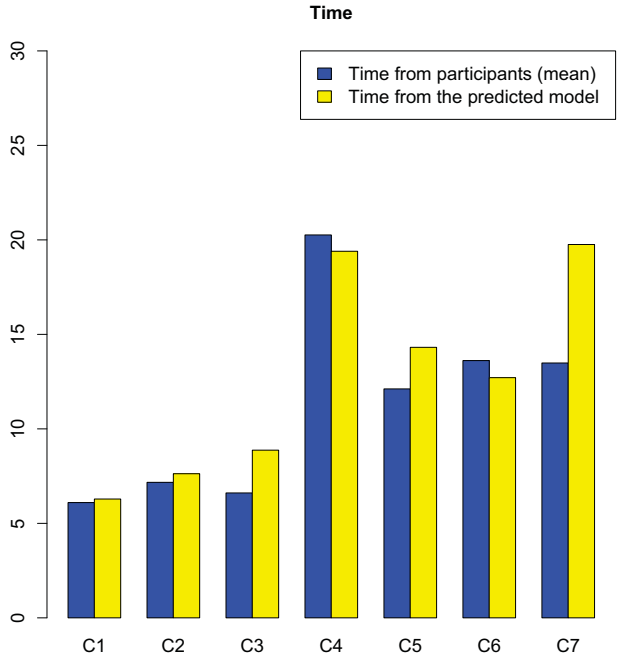

Figure 15: The time from the participants and the time from the predicted model where the correctness of the answers is set to 100 %

contrast to this the study is limited to the small-size dataset that consists of 7 components due to the limited time of the study session. Also the study is limited to the 4 component level metrics. However, the appropriate statistical analysis is performed in the study and some well-fitting models for the prediction of the effort required to understand a component are obtained. We plan to increase the number of studied components and to investigate more metrics in our future work.

Construct validity.

The construct validity is the degree to which the independent and the dependent variables are accurately measured by the appropriated instruments.

Regarding the dependent variables the percentage of the correct answers is measured through answering of the predefined set of questions, while the time variable is measured by entering the start and the stop time from the participants right before they start and right after they finish studying the given component. The percentage of the correct answers is calculated by checking if the answers from the participants are correct or incorrect. Therefore there are no threats to its accuracy. In contrast to this, the participants could have forgotten to write the time right before they start and right after they finish studying the components which represents a threat to the accuracy of the time variable. In order to reduce that threat we wrote a reminder before each component to remind the participants to write the stop time in the previously studied component if they forgot it and the start time for the given component they intend to study as the next one.

Independent variables that represent the component level metrics are calculated with the help of the tool ObjectAid

UML Explorer[6]. The dependencies between the source code classes are visualized in the tool and were then counted manually. There is only a very low threat to validity that the metrics calculations are not valid.

Internal validity.

The internal validity is the degree to which conclusions can be drawn about cause-effect of independent variables on the dependent variables. We dealt with the following issues:

- **Differences among subjects.** The subjects experience has approximately the same degree with regard to programming since most of them have at least medium experience in that area.
- **Accuracy of subject responses.** The threats for the accuracy of the variables time and the percentage of the correct answers are discussed in Section 5.
- **Other important factors.** Influence among subjects could not really be controlled. However, the study was carried out under supervision of a human observer. As no interactions between the participants have been observed, we assume that this potential threat did not affect the validity of the study.

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 following facts are identified:

- **Components and classes that are used.** For the purpose of the study we used small-size dataset that consists of 7 components of the Soomla Android store system due to the limited time of the study session. The number of studied components can be increased in replications of the study in order to be able to generalize the results. Another threat to external validity is the size of the classes in a component. As it is mentioned in Section 3.2.2 there are no big deviations in the sizes of the classes in the Soomla system, i.e. each of them provides one or a couple of similar functionalities. In the general case, there could be some classes that are much bigger than other classes in the system and in that case the number of classes in a component will not be an appropriate measure of its size. This case actually is not in accordance with good design principles, i.e. the big classes should be divided into smaller classes that encompass one or a couple of similar functionalities but this case can be examined in terms of which deviations in the classes' size are acceptable and do not violate the performed analysis.
- **Subjects.** In order to solve the difficulties about obtaining well-qualified subjects we used the master students at Faculty of Computer Science of the University of Vienna. While it is possible to show that they have substantial experience (and also industrial backgrounds), we are aware that more empirical studies with professionals need to be carried out in order to generalize the results.

[6]The ObjectAid UML Explorer is an agile and lightweight code visualization tool for the Eclipse IDE. It shows a Java source code and libraries in live UML class and sequence diagrams that automatically update as the code changes (www.objectaid.com).

6. CONCLUSIONS AND FUTURE WORK

In this paper we present the results obtained from a study we carried out to examine the relationships between four component level metrics that describe the components' size, coupling and complexity in terms of the number of classes they have and the classes' relationships on one side and the effort required to understand a component on the other side. The effort required to understand a component is measured through the time that the participants spent on studying each of the components. The study was conducted using the Soomla Android store system with 7 components in the architecture. Correlation, collinearity and multivariate regression analysis were performed. The results of the analysis show statistically significant correlation between 3 of the metrics (number of classes, number of incoming dependencies, and number of internal dependencies) on one side and the time obtained from the participants of the study on the other side. In a multivariate regression analysis we obtained 3 reasonably well-fitting models that can be used to estimate the effort required to understand a component measured though the time spent on studying it. In our future work we plan to study more components and to investigate more metrics and their relationships to the understandability of components and architectural component models.

Acknowledgement

This work was supported by the Austrian Science Fund (FWF), Project: P24345-N23. We thank Dr. Nina Seniitschnig 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.

7. REFERENCES

[1] Cross Validation techniques in R: A brief overview of some methods, packages, and functions for assessing prediction models.

[2] E. B. Allen, S. Gottipati, and R. Govindarajan. Measuring size, complexity, and coupling of hypergraph abstractions of software: An information-theory approach. *Software Quality Control*, 15(2):179–212, June 2007.

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

[4] D. A. Belsley, E. Kuh, and R. E. Welsch. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity (Wiley Series in Probability and Statistics)*. Wiley-Interscience.

[5] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos. Graph-based analysis and prediction for software evolution. In *ICSE'12*, pages 419–429, 2012.

[6] 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.

[7] P. Dalgaard. *Introductory Statistics with R*. Springer, Jan. 2004.

[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] J. J. Faraway. *Practical Regression and Anova using R*. July 2002.

[10] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. Softw. Eng.*, 26(8):797–814, Aug. 2000.

[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] V. Gupta and J. K. Chhabra. Package level cohesion measurement in object-oriented software. *J. Braz. Comp. Soc.*, 18(3):251–266, 2012.

[13] Z. Haohua, Z. Hai, C. Wei, and A. Jun. The method for measuring large-scale object-oriented software system. In *Proceedings of the 6th international conference on Fuzzy systems and knowledge discovery - Volume 3*, FSKD'09, pages 603–606, Piscataway, NJ, USA, 2009. IEEE Press.

[14] J. Hwa, S. Lee, and Y. R. Kwon. Hierarchical understandability assessment model for large-scale OO system. In *Proceedings of the 2009 16th Asia-Pacific Software Engineering Conference*, APSEC '09, pages 11–18, Washington, DC, USA, 2009. IEEE Computer Society.

[15] A. Kanjilal, S. Sengupta, and S. Bhattacharya. CAG: A Component Architecture Graph. In *TENCON, IEEE Region 10 International Conference*, 2008.

[16] 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.

[17] M. Kobayashi and S. Sakata. Mallows' cp criterion and unbiasedness of model selection. *Journal of Econometrics*, (3):385–395.

[18] M. Lindvall, R. Tesoriero, and P. Costa. Avoiding architectural degeneration: An evaluation process for software architecture. In *Proceedings of the 8th International Symposium on Software Metrics*, METRICS '02, pages 77–, Washington, DC, USA, 2002. IEEE Computer Society.

[19] Y. Ma, K. He, D. Du, J. Liu, and Y. Yan. A complexity metrics set for large-scale object-oriented software systems. In *Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, CIT '06, pages 189–, Washington, DC, USA, 2006. IEEE Computer Society.

[20] Y. Ma, K. He, B. Li, J. Liu, and X.-Y. Zhou. A hybrid set of complexity metrics for large-scale object-oriented software systems. *J. Comput. Sci. Technol.*, 25(6):1184–1201, 2010.

[21] Y. K. Malaiya and J. Denton. Module size distribution and defect density. In *Proceedings of the 11th International Symposium on Software Reliability Engineering*, ISSRE '00, pages 62–, Washington, DC, USA, 2000. IEEE Computer Society.

[22] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[23] P. Mohagheghi, R. Conradi, O. M. Killi, and H. Schwarz. An empirical study of software reuse vs. defect-density and stability. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 282–292, Washington, DC, USA, 2004. IEEE Computer Society.

[24] 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.

[25] 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.

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

[27] S. Sarkar, A. C. Kak, and G. M. Rama. Metrics for measuring the quality of modularization of large-scale object-oriented software. *IEEE Trans. Softw. Eng.*, 34(5):700–720, Sept. 2008.

[28] A. Sharma, P. S. Grover, and R. Kumar. Dependency analysis for component-based software systems. *SIGSOFT Softw. Eng. Notes*, 34(4):1–6, July 2009.

[29] G. Wei, X. Zhong-Wei, and X. Ren-Zuo. Metrics of graph abstraction for component-based software architecture. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 07*, CSIE '09, pages 518–522, Washington, DC, USA, 2009. IEEE Computer Society.

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

[31] L. Yu, K. Chen, and S. Ramaswamy. Multiple-parameter coupling metrics for layered component-based software. *Software Quality Journal*, 17(1):5–24, 2009.

[32] T. Zhou, B. Xu, L. Shi, Y. Zhou, and L. Chen. Measuring package cohesion based on context. In *Proceedings of the IEEE International Workshop on Semantic Computing and Systems*, WSCS '08, pages 127–132, Washington, DC, USA, 2008. IEEE Computer Society.