Controlled Experiment on the Supportive Effect of Architectural Component Diagrams for Design Understanding of Novice Architects

Thomas Haitzer and Uwe Zdun

Research Group Software Architecture
University of Vienna
Austria
{thomas.haitzer | uwe.zdun}@univie.ac.at

Abstract. Today, architectural component models are often used as a central view of architecture descriptions. So far, however, only a very few rigorous empirical studies relating to the use of component models in architectural descriptions of software systems have been conducted. In this paper, we present the results of a controlled experiment regarding the supportive effect of architectural component diagrams for design understandability. In particular, the goal of the experiment was to determine whether architectural component diagrams, provided in addition to a non-trivial software system's source code, have a supportive effect on the ability of novice architects to answer design and architecture related questions about that system. Our study provides initial evidence that architectural component diagrams have a supportive effect for understanding the software design and architecture, if a direct link from the component diagram's elements to the problem that requires understanding can be made. If such a direct link cannot be made, we found evidence that it should not be assumed that architectural component diagrams help in design understanding, for instance only by providing a big picture view or some general kind of orientation.

Keywords: Software Architecture, Architectural Component Diagrams, Design and Architecture Understanding, Empirical Study, Controlled Experiment

1 Introduction

Today a software architecture description is usually comprised of multiple views [5, 19, 20]. The component and connector model (or component model for short) of an architecture is a view that is often considered to contain the most significant architectural information [5]. This view deals with the components, which are units of runtime computation or data-storage, and the connectors which are the interaction mechanisms between components [5, 29]. An architectural component model is a high-level abstraction of the entities in the source code of the software system, as the software architecture concerns only the major design decisions about a software system, and abstracts from irrelevant details [21].

While much research work has been done in component-related research areas such as modelling languages for component and connector models, component implementation technologies, component composition, and the formal semantics of components, only a very few rigorous empirical studies relating to the use of component models in architectural descriptions of software systems have been conducted. Such foundational research is however essential to provide guidelines and tools to software architects, based on sound evidence, to help them understand how to design component models that are appropriate for the architectural understanding of a software system.

In this paper, we present the results of a controlled experiment regarding the supportive effect of architectural component diagrams for design understandability. In particular, the goal of the experiment was to determine whether architectural component diagrams, provided in addition to a non-trivial software system's source code, have a supportive effect on the ability of novice architects to answer design and architecture related questions about that system. This goal is interesting to study, as today it is unclear whether component diagrams alone are sufficient to help architects to understand complex architectural relationships in a given system in a better way than just studying the source code of that system. While the literature suggests a supportive effects of component diagrams (see e.g. [5, 31]) for design understanding, there is little empirical evidence so far.

In addition, many existing approaches seem to assume seasoned architects as their main target group. Assuming that component diagrams alone are a useful help to gain a better architectural understanding of a system, as some of the software architecture literature suggests, it is unclear whether this effect can also be observed for novice architects. As software architecture has the goal to convey the big picture of a software system and novices who start on a new project especially require help to gain such a big picture quickly, it is highly interesting whether there is indeed a supportive effect on design understanding for them. Hence, we particularly focus on novice architects.

The experiment presented in this paper studies the experiment goal by letting 60 students with medium programming experience answer seven questions about the design and architecture of a given software system (the computer game FreeCol). One half of the participants, the control group, received the source code of that system as the main source of information, while the other half of the participants, the experiment group, additionally received architectural component diagrams for FreeCol. By showing that the quality of the answers improves for certain questions, our study provides initial evidence on how architectural component diagrams help in understanding the design and architecture of software systems. The results indicate that architectural component diagrams are especially useful if a direct link from the component diagram's elements to the problem that requires understanding can be made and that they have in such cases indeed a supportive effect for software design and architecture understanding. In contrast, if no such direct link can be made, we found evidence that it should not be assumed that architectural component diagrams help in design understanding, for instance only by providing a big picture view or some general kind of orientation.

This paper is organized as follows: In Section 2 we briefly discuss the related work. Next, in Section 3 we introduce our experiment, including the goal, the hypotheses, the parameters and variables, the experimental design, and the execution. Section 4 describes the statistical analysis and the testing of the hypotheses. Section 5 provides the validity evaluation. Finally, Section 6 concludes and discusses possible future research directions.

2 Related Work

The general notion of empirical studies in software architecture has been studied by Falessi et al. [8]. They conclude from their study that a greater synergy between empirical software engineering and software architecture would support the emergence of a body of knowledge consisting of more widely accepted and well-formed theories on software architecture and the empirical maturation of the software architecture area.

Only a few of the empirical studies in the area of software architecture are directly related to architecture design or design understanding. Boucke et al. [4] introduce an approach that explicitly supports compositions of models, together with relations among models in an architecture description language. In an empirical study they show that their approach reduces the number of manually specified elements and manual changes.

van Heesch et al. study in two surveys the reasoning process of architects, one with students [16] and one with professionals [18]. A related study performs a controlled experiment about the supportive effect of patterns in architecture decision recovery [17].

Many empirical studies in the field of software architecture study other aspects, like quality aspects or other views. For instance, a number of studies related to evaluating architectures have been conducted. Barbar et al. [2] performed an empirical study aiming at understanding the different factors involved in evaluating architectures in industry. The influence of software visualization on source code comprehensibility was studied by Umphress et al. [35] based on control structure diagrams and complexity profile graphs. Biffl. et al. [3] study the impact of experience and team size on the quality of scenarios for architecture evaluation. A number of empirical studies aim at better understanding the relation of architecture and requirements [11, 26]. Various empirical studies relating architecture to certain qualities or metrics have been conducted. For example, Hansen et al. study the relation of product quality metrics and architecture metrics [15].

A number of papers focus on the comprehension of UML diagrams. Some focus on dynamic models [28], while others focus on specific diagrams or models like sequence diagrams [13], state charts [7], or class diagrams [30]. The influence of the level of detail in class and sequence diagrams on the maintainability of software has been studied by Fernández-Sáez et al. [10]. These papers focus on factors that influence the understandability of the diagrams itself, while we focus on the effects of component diagrams on the architecture understanding of novice architects.

Even though we found no rigorous empirical studies of architectural component model understandability so far, aspects like reuse or fault density of components have been studied empirically before. Fenton and Ohlsson have studied the relations of fault density and component size in a large telecom system [9]. Mohagheghi et al. provide a study comparing software reuse with defect density and stability [27]. Their study is based on historical data on defects, modification rate, and software size. Malaiya and Denton provide an analysis of a number of studies and identify the component partitioning and implementation as influencing, competing factors to determine the "optimal" component size with regard to fault density [24]. Graves et al. have studied the software change history of components to create a fault prediction model [14]. Our experiment and these studies have in common, that they make a link between component models and software quality, but in contrast to our experiment they only study aspects that can solely be studied using the software systems and their historical data. In contrast, we consider the (novice) architect's perception of understandability as well as expert opinions on their results. In addition, in those other studies components are understood as implemented software modules, rather than architectural abstractions.

A number of approaches suggest that other aids are needed to gain a better understanding of the design or architecture, such as architectural views [5, 19, 20] or architectural decision models [23, 34, 38], which would contain or augment component models.

Both research directions only focus on complementing component models with additional knowledge, but do not research on the effects of the component models on the understandability of a software architecture or design. Other literature suggests that it might be hard to understand the source code only with models, and traceability links [1] between components and code are needed to make the connection [12].

3 Experiment Description

For the design of the experiment we followed the guidelines by Kitchenham et al. [22] and Wohlin et al. [37]. In our experiments, the guidelines by Kitchenham et al. were primarily used in the planning phase of the experiments, while the advice by Wohlin et al. was used as a reference for the analysis and interpretation of the results.

3.1 Goal and hypotheses

The goal of the experiment was to determine whether architectural component diagrams, provided in addition to a non-trivial software system's source code, have a supportive effect on the ability of novice architects to answer questions about the design and architecture of that system. Depending on the question asked, the guidance or help provided by architectural component diagrams can vary greatly. The two extreme cases are that component diagrams readily provide the answer without any need to study other information (like the source code) and that component diagrams provide no clue for answering the question. Intentionally we left out these two extreme cases and studied the shades of grey in between. In particular, we further distinguished the following three types of questions in our experiment:

– $QT1$: A question about the software system's design and architecture for which the component diagrams provide some guidance or help, but the information in the component diagrams alone is not enough to answer the question fully.
– $QT2$: A question about the software system's design and architecture for which the component diagrams provide some guidance or help, but the same information is easily visible from the source code.
– $QT3$: A question about the software system's design and architecture for which the component diagrams provide no direct guidance or help, only vague orientation in related components and connectors; digging in the source code is required for answering the question.

Hypotheses: We postulate the following three hypotheses about the effects of architectural component diagrams (in addition to the source code) on the quality of answers that novice architects provide to questions about a software system's design and architecture.

– In case of $QT1$, i.e. if the component diagrams provide architectural guidance for answering the question,
  • the null hypothesis is that the quality does not improve, $H_{0\_QT1} : \mu \leq \mu_0$;
  • the alternative hypothesis is that the quality improves, $H_{QT1} : \mu > \mu_0$.
– In case of $QT2$, i.e. if the component diagrams provide architectural guidance for answering the question, but the same information is visible from the source code,
  • the null hypothesis is that the quality does not improve, $H_{0\_QT2} : \mu \leq \mu_0$;

  - the alternative hypothesis is that the quality improves, $H_{QT2} : \mu > \mu_0$.
- In case of $QT3$, i.e. if the component diagrams provide no direct guidance or help, only vague orientation in related components and connectors,
  - the null hypothesis is that the quality does not improve, $H_{0\_QT3} : \mu \leq \mu_0$;
  - the alternative hypothesis is that the quality improves, $H_{QT3} : \mu > \mu_0$.

Expectations: Our expectations for the three hypotheses are:

- For design questions of type $QT1$, we *expect that the null hypothesis can be rejected*. That is, component diagrams have a supportive effect on the answers that novice architects provide to questions about a software system's design and architecture, if the component diagrams provide architectural guidance for answering the question.
- For design questions of type $QT2$, we expect that the *null hypothesis can not be rejected*. That is, component diagrams are helpful, but that novice architects with medium software development experience are able to see the same information in the source code, if it is easily visible. However, this expectation might be wrong as possibly the visual information in the component diagrams might be more readily accessible to novice architects than the easily visible information in the source code.
- For design questions of type $QT3$, we expect that the *null hypothesis can not be rejected*, as there is no direct relation between the question and the additional information provided by the component diagrams. However, this might be wrong as component diagrams might have an indirect supportive effect, for instance by providing some kind of general orientation that helps in answering this type of questions.

3.2 Parameters and variables

Dependent variable One dependent variable was observed during the experiment, as shown in Table 1: the quality of the answer to the design question. The quality of the answers was assessed by three independent software architecture researchers with multiple years of practical software development and architecture experience (later also referred to as analysts) using an interval scale, ranging from 0 (worst) to 10 (best). The interval scale nature of the rating system was explained to the analysts before their analysis (i.e., that equal distances between the points on the scale can be assumed), as this is important for applying parametric statistical tests [33]. The analysts were assigned per question, and each analyst rated each of the assigned questions completely in both experiment and control group, to make sure that each question is homogeneously assessed. Two analysts rated two of the questions, and one analyst rated three of the questions. Each of the analysts studied the two software systems used in the experiments before their analysis in depth and reference answers were created before the evaluation to ensure fair evaluation. The ratings were left to the analysts' own experience and interpretation, but they were asked to specifically take the displayed architectural understanding into account. The participants of the experiments were also reminded before the beginning of the experiments that they should focus on the architectural dimension of the questions.

Independent variables Table 1 also shows other variables that could influence the dependent variables. They concern characteristics and previous experiences of the participants.

Type	Description	Scale Type	Unit	Range
Dependent Variable	Quality of the answer to design question	Interval	Points	0 (worst) to 10 (best)
Independent Variable	Group	Nominal	N/A	Possible values: *experiment group*, *control group*
	Programming experience	Ordinal	Years	4 classes: 0-1, 1-3, 3-6, >6
	Commercial programming experience in industry	Ordinal	Years	4 classes: 0, 0-1, 1-3, >3
	Experience in programming computer games	Ordinal	Years	4 classes: 0, 0-1, 1-3, >3

Table 1: Observed Variables

3.3 Experiment design

To test the hypotheses, we conducted the experiment in the context of the Software Architecture course at the Faculty of Computer Science, University of Vienna in spring 2012.

Subjects The subjects of the experiment are 60 students of the Software Architecture course. The subjects were randomly assigned into two equally sized groups of 30 students: experiment group and control group.

Objects The basis of the experiment is the source code of the Freecol computer game[1], an open source version of the classic computer game Colonization (a turn-based strategy game) with multi-player support, implemented in Java. Both experiment group and control group were provided with access to the complete source code of Freecol. In order to avoid any bias caused by complex Integrated Development Environments (IDEs) or code editors, source code access was only provided using the Browser-based code navigation tool that is integrated with our locally hosted installation of Gitorious[2].

Instrumentation The participants of both groups received the following materials: The Browser-based access to the source code of Freecol was provided in a Lab environment on prepared computers. All other materials were provided on paper. The participants received a questionnaire about the independent variables regarding the participants' experiences. Both groups also received 7 different questions about the design and architecture of Freecol (see below). In addition, the experiment group received an additional document with 6 UML component diagrams showing: a FreeCol Architecture Overview, the FreeCol Server Architecture, the FreeCol Client Architecture, a Detailed View: FreeCol Server - Control, the FreeCol MetaServer Architecture, and a Detailed View: FreeCol Commons. The component diagrams have been created in an architectural reconstruction of FreeCol that took place before the experiment and was independent of the experiment.

In the experiment we have tested 7 different questions about the design and architecture of Freecol. The questions have been confirmed by the independent analysts as being relevant questions for understanding Freecol's design and architecture. The questions and their classifications are shown in Table 2. We have classified the questions into the question types from Section 3.1 as follows:

[1] See `http://www.freecol.org/`.
[2] See `http://www.gitorious.org/`.

- $QT1$: For 3 questions ($Q1$, $Q5$, $Q7$) the component diagrams provide direct guidance or help to better understand Freecol's design and architecture, but the information in the component diagrams alone is not enough to answer the question fully.
- $QT2$: For 1 question ($Q2$) the answer can be deduced both from the component diagrams and the source code organization (in packages) alike.
- $QT3$: For 3 questions ($Q3$, $Q4$, $Q6$) the component diagrams provide no obvious information, only vague orientation, and digging in the source code is required to answer the question.

We asked 3 questions of type $QT1$ and 3 questions of type $QT3$, so clearly those two question types are our main focus. We also checked $QT2$, but only once, as it is a rather seldom occurring option somewhat in between $QT1$ and $QT3$ that some design aspects are directly visible from the source code organization. To illustrate the difference between the two extremes in our experiment, $QT1$ and $QT3$, and let us briefly explain the difference in the level of detail modelled:

- *Example for Type $QT3$:* For question $Q3$ of type $QT3$ there is only a component *AI* visible in the FreeCol Server Architecture model. It has a single connector with the interface *AIPlayer* to the *Model* component. This provides only *vague orientation* for answering question $Q3$, as it enables the participants to know that AI concerns are implemented in the server packages and that there is a link to the model classes, but it does not provide details for answering the question.
- *Example for Type $QT1$:* For question $Q1$ of type $QT1$ there are significantly more details and links to all important aspects of the question in the component diagrams. First of all in the FreeCol Client Architecture model, we can see a component *Controller* that is linked through a connector with an interface *GameControl* to the *Actions* component and through another connector with an interface *UpdateHandler* to the *GUI* component. This enables participants to understand how GUI Actions use *InGameController* (and another class *ConnectController*) to perform model, game, etc. updates using basic controls. It also makes it easy to find various basic control tasks in the game's client, which can then easily be found in the source code. The *Controller* also has a connector to a port with the *Model* interface, which links to details in the Commons and Server Architecture component models. In the server, the *Model* component is linked to another *Control* component which is modelled in a detailed view, the FreeCol Server: Control component diagram. This enables participants to understand (1) the client-server relationship for control and the synchronization through models and (2) the event handling for changes through model messages.

Clearly, the level of detail for question type $QT1$ is much higher. We hope this example helps to illustrate what we mean by *"providing direct guidance or help"* for $QT1$ in contrast to *"vague orientation"* for $QT3$.

3.4 Execution

The experiment was executed in the context of the Software Architecture course at the Faculty of Computer Science, University of Vienna in the summer semester 2012/2013. Before the experiment took place, the participants were randomly assigned to experiment group and control group. Each of the two groups consisted of 30 participants (total participant number: 60).

ID	Type	Question	Classification Details
Q1	QT1	Explain the role of the class "InGameController" in the Package "net.sf.freecol.client.control". What is its purpose?	Component diagrams provide detailed orientation through related components and connectors, and also hint at interesting architectural concerns not easily seen from the source code. Connection to the source code must be made for providing the full answer.
Q2	QT2	How many and which independent executable programs belong to this game?	Component diagrams provide detailed orientation. The answer can be deduced from component model, but also from the package structure in the source code.
Q3	QT3	Explain how the computer players (AI) are integrated into the game. Which classes are responsible for the integration and implementation of the AI players? In which of the executable program(s) (see Q2) do they run?	Component diagrams provide vague orientation through components. The source code is the main source of information.
Q4	QT3	What is the role of the class "DOMMessage" in the "net.sf.freecol.common.networking" Package? How is it used in the game?	Component diagrams provide no details for answering the question, only vague orientation. The source code is the main source for getting the required information.
Q5	QT1	What is the role of classes in the package "net.sf.freecol.metaserver"?	Component diagrams provide detailed orientation through related components and connectors. Connection to the source code must be made for providing the full answer.
Q6	QT3	What are the roles of the classes in the packages "net.sf.freecol.server.model", "net.sf.freecol.client.control", "net.sf.freecol.common.model"? How are they related to each other?	Component diagrams are useful for basic orientation, but not for answering the question. The source code is the main source of information.
Q7	QT1	In order to show a consistent game state to all players, the programs of the different players must be updated regularly. How and by which classes is this mechanism realized? Sketch the control flow from one class (or object) to the next one.	Component diagrams show related components and connectors, and a few details helpful for answering the question. Component diagrams also hint at the architectural big picture not easily seen only from the source code.

Table 2: Questions and Classification of Questions

Figure 1 shows the previous experience of the participants for the control group and the experiment group. In particular, the figure shows the programming experience of the participants, which is quite comparable in the two groups, with slightly more participants with longer experience in the experiment group. The industry programming experience is low in both groups, with a few participants with 1, 1-3, or even more than 3 years of industry experience in both groups. Finally, the very few participants with game programming experience are equally present in both groups.

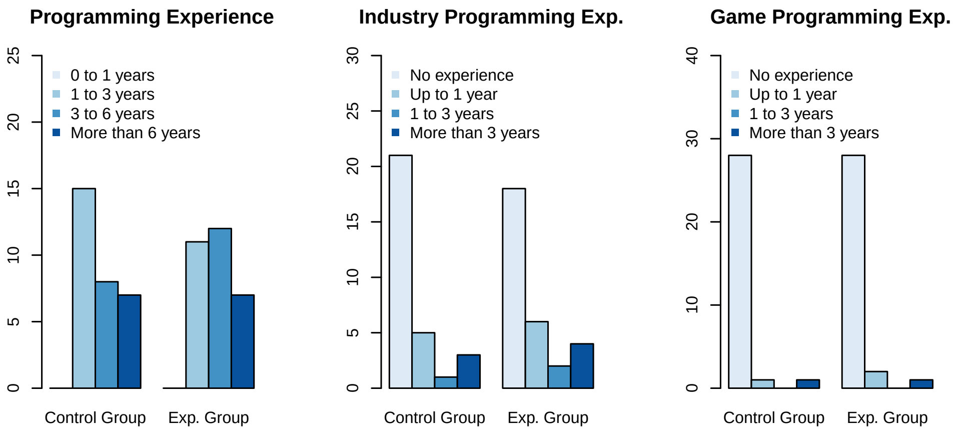

Fig. 1: Experience of the Participants

Before the experiment started, the materials explained in Section 3.3 were handed out to the participants and the tasks were briefly explained to both groups. After 15 minutes of introduction, the participants were given time to fill out the questionnaire about their experiences. After all participants were ready, the answering of the questions started. The answers were provided by the participants on paper. This main phase of the experiment lasted for two hours.

The data collection was performed as planned in the design. No participants dropped out and no deviations from the study design occurred.

The experiment took place in a controlled environment. The experiment was conducted for both groups in different rooms, equipped with computers to which the participants had logins. At least one experimenter was present in each room during the whole experiment time to assure that participants behaved as expected. After the experiment, all materials were collected by the experimenters before any of the participants left the room. There were no situations in which participants behaved unexpectedly.

4 Analysis

4.1 Descriptive Statistics

Figure 2 shows the medians and means for the quality of the answers given to the seven questions $Q1$–$Q7$ for both control group and experiment group (the values can also be seen below in Table 4). As can be seen, the medians and means for questions of type $QT1$ ($Q1$, $Q5$, $Q7$) are always higher in the experiment group than those in the control group.

For the question of type $QT2$ ($Q2$) the control group yields a slightly better result. The medians and means for questions of type $QT3$ ($Q3$, $Q4$, $Q6$) of the experiment group show the same or slightly better results than those of the control group.

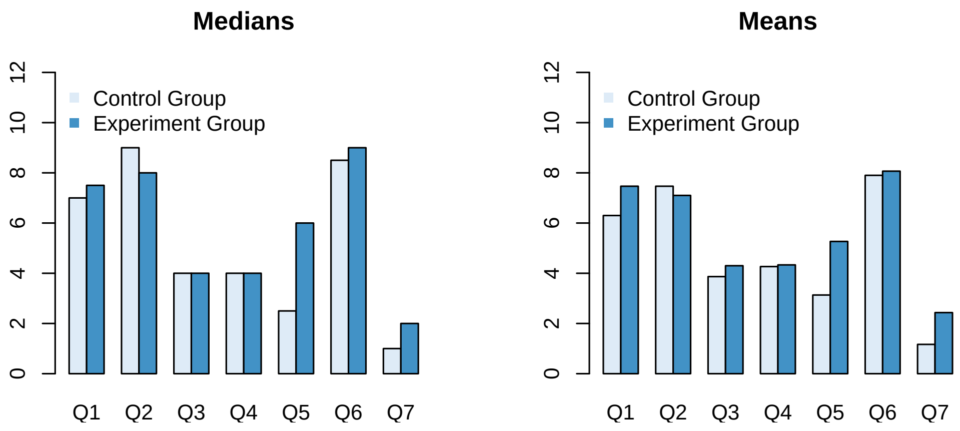

Fig. 2: Medians and means for the seven questions

4.2 Data Set Reduction

The deviations from the means for the ratings of all questions are in a corridor that roughly corresponds to our previous experiences from other exercises with participants in our courses. Hence, we did not want to exclude individual participants from the data set, as excluding data points would have introduced a potential vulnerability for the study results.

An interesting outlier in the medians and means for the seven questions is Question $Q7$, where both groups performed rather poorly. Hence, we studied the answers for this question in depth to understand whether it is necessary to exclude Question $Q7$ from the further analysis, for instance because it was too hard to answer or simply because the participants did not have enough time for answering the question (which was the last question). First we checked the protocols of the experiment and most participants have finished before the end of the 2 hours slot, so the limited time frame does not seem to be the cause of the poor results. To study whether the question was too difficult, we did an in depth study of answers without knowledge whether the individual answers were from the control group or the experiment group. The results are: Indeed, Question $Q7$ seems to be a difficult question that requires complex design and architecture understanding and making connections across multiple parts of the FreeCol system's design and architecture. Most participants failed and reached 0-3 points. Very few participants are in the middle ranks of 4-6 points. Only 6 out of the 60 participants managed to provide a sufficient answer to the question (with a score > 6 points). As this means that 10% of the participants were able to answer the question sufficiently, it does not seem impossible for novice architects to answer Question $Q7$, just difficult. Hence, we decided to not exclude the question from further analysis but rather view it as a case to study a difficult question of type $QT1$.

4.3 Hypotheses testing

Testing the normality of the data As a first step in analysing the data, we tested the normality of the data by applying the Shapiro-Wilk test [32], in order to see whether we can apply parametric tests like the t-test that assume the normal distribution of the analysed data. The null hypothesis H_0 for the Shapiro-Wilk test states that the input data is normally distributed. H_0 is tested at the significant level of $\alpha = 0.05$ (i.e., the level of confidence is 95%). That is, if the calculated p-value is lower than 0.05 the null hypothesis is rejected and the input data is not normally distributed. If the p-value is higher than 0.05, we can not reject the null hypothesis that the data is normally distributed.

Group	**N**	**Shapiro-Wilk test p-value**						
		Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**
Control Group	30	0.06505	6.528e-05	0.07345	0.0852	0.005865	7.255e-05	1.362e-06
Experiment Group	30	0.01998	9.035e-05	0.04852	0.3658	0.0002576	3.028e-05	0.0007023

Table 3: Results of the Shapiro-Wilk normality test

In the Table 3 the p-values for the Shapiro-Wilk normality test for the seven questions $Q1$–$Q7$ for both control group and experiment group are shown. As can be seen, most questions do not have a normal distribution (i.e., hypothesis H_0 is rejected). Some other questions show a p-value right above 0.05 which means a very weak tendency of being normally distributed. In order to test the normality of the variables with a p-value above 0.05, we graphically examined how well these variables fit the normal distribution using the normal Q-Q plot. Q-Q plot is a graphical method for comparing two probability distributions by plotting their quantiles against each other [36]. Normal Q-Q plot is a method for graphically comparing the probability distribution of the given data sample with the normal distribution. While some of the resulting plots fit the normal distribution pretty well, for none of the questions both control group and experiment group showed a strong tendency to be normally distributed. Based on these considerations of normality, we decided to pursue non-parametric statistical tests with our data.

Comparison of the means between two variables To compare the means of the variable for the control group and experiment group of a question, we applied the Wilcoxon rank-sum test [25]. The Wilcoxon rank-sum test is a non-parametric test for assessing whether one of two data samples of independent observations is stochastically greater than the other. The null hypothesis of the one-tailed Wilcoxon test (appropriate for the hypotheses in our experiment) is that the means of the first variable's distribution is less than or equal to the means of the second variable's distribution,so that we can write $H_0 : A \leq B$. The Wilcoxon rank-sum test tries to find a location shift in the distributions, i.e., the difference in means of two distributions. The corresponding alternative hypothesis H_A could be written as $H_A : A > B$. If a p-value for the test is smaller than 0.05 (level of significance), the null hypothesis is rejected and the distributions are shifted. If a p-value is larger than 0.05, the null hypothesis can not be rejected, and we can not claim that there is a shift between the two distributions.

In the Table 4 the p-values for the Wilcoxon rank-sum test are shown, together with means and medians. The raw material for these results can be downloaded from `https://swa.univie.ac.at/CDE`. Based on the obtained p-values, we can assess that the following distributions show a statistically significant shift between each other: $Q1$, $Q5$, and $Q7$. None of the other variables shows a statistically significant shift.

ID	Control Group: Mean	Experiment Group: Mean	Control Group: Median	Experiment Group: Median	p-value
Q1	6.3	7.466667	7	7.5	0.02021
Q2	7.466667	7.1	9	8	0.4818
Q3	3.866667	4.3	4	4	0.263
Q4	4.266667	4.333333	4	4	0.494
Q5	3.133333	5.266667	2.5	6	0.01512
Q6	7.9	8.066667	8.5	9	0.2212
Q7	1.166667	2.433333	1	2	0.006899

Table 4: Results of the Wilcoxon rank-sum test

Testing Hypothesis H_{QT1} In our experiment, we have introduced 3 questions related to H_{QT1}: $Q1$, $Q5$, and $Q7$. Each of the three questions shows a significant location shift in their distributions, and in each of them the experiment group shows better results than the control group in their means and medians. This provides evidence that H_{0_QT1} can be rejected. That is, indeed there is evidence that augmenting the source code with architectural component diagrams *improves the quality* of answers that novice architects provide to questions about a software system's design and architecture, if the component diagrams provide architectural guidance for answering the question.

It is interesting to note that the difficult Question $Q7$ shows the same result (even with the highest significance level) as Questions $Q1$ and $Q5$ (of medium difficulty). While many of the participants in the experiment group failed as well, all but one of the sufficient answers were in the experiment group. This result seems to indicate that component diagrams can be especially helpful for problems that require making complex design and architecture connections across multiple parts of the system.

Testing Hypothesis H_{QT2} In our experiment, we have introduced 1 question related to H_{QT2}: $Q2$. For this question we can observe higher means and medians for the control group than for the experiment group, however the location shift is not statistically significant. Therefore H_{0_QT2} can *not* be rejected. As expected, there is *no evidence* that augmenting the source code with architectural component diagrams does improve the quality of answers that novice architects provide to questions about a software system's design and architecture, *if the component diagrams provide architectural guidance for answering the question, but the same information is visible from the source code*.

Testing Hypothesis H_{QT3} In our experiment, we have introduced 3 questions related to H_{QT3}: $Q3$, $Q4$, and $Q6$. The medians and means of the experiment group show the same or slightly better results than those of the control group. None of the three questions shows a significant location shift in their distributions so H_{0_QT3} *can not be rejected*. As expected, there is *no evidence* that augmenting the source code with architectural component diagrams does improve the quality of answers that novice architects provide to questions about a software system's architecture, *if the component diagrams provide no direct guidance or help, only vague orientation in related components and connectors*.

5 Validity evaluation

Several levels of validity have to be considered in this experiment. We consider the classification scheme for validity in experiments by Cook and Campbell [6].

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

- The subjects' experiences in the two groups have approximately the same degree with regard to programming, industrial, and game programming experience. Of course, a certain differences in experience between the two groups can not be entirely excluded.
- All subjects' have at least medium programming experiences and have passed several courses on programming and design at our university. Hence, we consider their responses as valid, keeping in mind that our goal was to analyse the supportive effects component diagrams have on novice architects.
- The experiment lasted about 2 hours so fatigue effects were not considered relevant.
- The experiment happened in a controlled environment in separate rooms under supervision of at least one experimenter. While it is not possible to completely exclude misbehaviour or interaction among participants, it is not very likely that misbehaviour or interactions have had a big influence on the outcomes of the experiment.
- Possibly the analysts could have been biased towards the experiment group in some way. We tried to exclude this threat to validity by not revealing to the analysts the identity of the participants or in which of the two groups they have participated. Hence, it is rather unlikely that this threat occurred.

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 greater the external validity, the more the results of an empirical study can be generalised to software engineering practice.

- We used students of our software architecture lecture as subjects. As discussed , they have medium programming and design experience, but limited professional experience. Hence, we believe them to be well representative for the target group of novice architects, but if and how the results can be translated to more experienced architects is open to future study. We plan to replicate the experiment with other target groups.
- The instrumentation and object in the experiment might have been unrealistic, not representative, or too simple to allow generalization. For instance, FreeCol as an open source game might be too simple or no representative software system for typical architectural studies. The component diagrams used might not be representative or unrealistic. Or the questions asked might not be typical design or architecture questions. All these considerations might impede the generalizability of our results. We do not think that this is the case as FreeCol is a widely used, non-trivial software system implemented in Java. The component diagrams were created in an architectural reconstruction effort that took place before the experiment and was independent of the experiment. The questions have been confirmed by the independent analysts as being relevant questions for understanding design and architecture of FreeCol.
- The experimenters could have biased the measurements of the independent variables. We mitigated this risk by assigning the quality ratings to independent analysts that had no knowledge about the goals of the experiment. Furthermore the analysts did not know the identities nor the groups of the participants.

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 (60 participants, 30 in each group). The size can be increased in replications of the study in order to reach normality of the obtained data. We plan to replicate the study with different systems and by engaging subjects who work in industry in our future work.

Construct validity The construct validity is the degree to which the independent and the dependent variables are accurately measured by their appropriated instruments. As only one object, the FreeCol implementation and associated component diagrams, was used in the experiment, there is the risk that the cause construct is under-represented. Possibly, the results could look different if multiple systems and sets of diagrams would be used for the recovery. We assume that the used system is representative for large and medium-size object-oriented systems. This threat, however, can not totally be ignored. Another potential threat to validity is that we only used one variable to measure the quality of answers. This does not allow cross-checking the results with different measures.

6 Conclusions

Our study provides initial evidence on how architectural component diagrams help in understanding the design and architecture of software systems. The results indicate that architectural component diagrams are especially useful if a direct link from the component diagram's elements to the problem that requires understanding can be made. Component diagrams seem to help in such cases to understand the bigger architectural connections that are hard to see from studying the source code alone and/or they provide orientation in the source code to understand such problems. However, there is a different situation for problems that are readily solvable by looking at the source code (like the question of type $QT2$ in our experiment) or for problems that are only vaguely linked to what is depicted in the component diagrams (like the question of type $QT3$ in our experiment). As expected, we found no evidence that architectural component diagrams help, for instance, just by providing a big picture view or providing some general kind of orientation.

We can conclude for the design of architectural component diagrams that they should be designed with specific important architectural problems in mind and that elements of the component diagrams should explicitly represent links to those problems. That is, components, connectors, and other model elements for providing an abstract understanding of the design that resolves the problem should be shown in the diagrams. The component models related to questions of type $QT1$ in our experiment achieve this by leaving out irrelevant details and showing high-level connections of system parts that are hard to reconstruct by just studying the low-level source code classes. It also seems to be important that these links from component models to the problem in focus are modelled in enough detail. Only vaguely showing a problem-related component in its context of other components and connectors that are not related to the problem is not enough.

It seems plausible, based on our results, that such details could also be provided through other architectural views or through architectural knowledge models. Further, it seems that making links to the source code is important for the supportive effect revealed in our study. Such links can be made explicit through traceability links. Hence, it also seems plausible that establishing traceability links between architectural component models and code might have a further supportive effect. We plan to study these aspects in further studies in our future work. From the combined results of this and future studies we plan to develop design guidelines for architectural component diagrams. Regarding generalizability, our results are strictly limited to the target group of novice architects with medium programming experience. We expect that similar results will also show for seasoned architects, but potentially they can make more use of vague information in architectural component diagrams. Again, we plan to investigate this in our future research.

Acknowledgement This work was partially supported by the Austrian Science Fund (FWF), Project: P24345-N23.

Bibliography

[1] IEEE Standard Glossary of Software Engineering Terminology. Tech. rep. (1990)

[2] Babar, M.A., Bass, L., Gorton, I.: Factors influencing industrial practices of software architecture evaluation: an empirical investigation. In: Proceedings of the International Conference on the Quality of Software Architectures. pp. 90–107. QoSA'07, Springer-Verlag, Berlin, Heidelberg (2007)

[3] Biffl, S., Ali Babar, M., Winkler, D.: Impact of experience and team size on the quality of scenarios for architecture evaluation. In: Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering. pp. 1–10. EASE'08, British Computer Society, Swinton, UK, UK (2008)

[4] Boucké, N., Weyns, D., Holvoet, T.: Composition of architectural models: Empirical analysis and language support. J. Syst. Softw. 83(11), 2108–2127 (Nov 2010)

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

[6] Cook, T.D., Campbell, D.T.: Quasi-Experimentation: Design and Analysis Issues for Field Settings. Houghton Mifflin (1979)

[7] Cruz-Lemus, J.A., Genero, M., Manso, M.E., Morasca, S., Piattini, M.: Assessing the understandability of uml statechart diagrams with composite states–a family of empirical studies. Empirical Softw. Engg. 14(6), 685–719 (Dec 2009)

[8] Falessi, D., Babar, M.A., Cantone, G., Kruchten, P.: Applying empirical software engineering to software architecture: challenges and lessons learned. Empirical Softw. Engg. 15(3), 250–276 (Jun 2010)

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

[10] Fernández-Sáez, A.M., Genero, M., Chaudron, M.R.V.: Does the level of detail of uml models affect the maintainability of source code? In: Proceedings of the 2011th international conference on Models in Software Engineering. pp. 134–148. MODELS'11 (2012)

[11] Ferrari, R., Miller, J.A., Madhavji, N.H.: A controlled experiment to assess the impact of system architectures on new system requirements. Requir. Eng. 15(2), 215–233 (Jun 2010)

[12] Galster, M.: Dependencies, traceability and consistency in software architecture: towards a view-based perspective. In: Proceedings of the 5th European Conference on Software Architecture: Companion Volume. ECSA '11, ACM (2011)

[13] Genero, M., Cruz-Lemus, J.A., Caivano, D., Abrahão, S., Insfran, E., Carsí, J.A.: Assessing the influence of stereotypes on the comprehension of uml sequence diagrams: A controlled experiment. In: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems. pp. 280–294. MoDELS '08, Springer-Verlag (2008)

[14] Graves, T.L., Karr, A.F., Marron, J.S., Siy, H.: Predicting fault incidence using software change history. IEEE Trans. Softw. Eng. 26(7), 653–661 (Jul 2000)

[15] Hansen, K.M., Jonasson, K., Neukirchen, H.: Controversy corner: An empirical study of software architectures' effect on product quality. J. Syst. Softw. 84(7), 1233–1243 (Jul 2011)

[16] van Heesch, U., Avgeriou, P.: Naive architecting - understanding the reasoning process of students: a descriptive survey. In: Proceedings of the 4th European conference on Software architecture. pp. 24–37. ECSA'10, Springer-Verlag, Berlin, Heidelberg (2010)

[17] van Heesch, U., Avgeriou, P., Zdun, U., Harrison, N.: The supportive effect of patterns in architecture decision recovery - a controlled experiment. Sci. Comput. Program. 77(5), 551–576 (May 2012)

[18] Heesch, U.v., Avgeriou, P.: Mature architecting - a survey about the reasoning process of professional architects. In: Proceedings of the 2011 Ninth Working IEEE/IFIP Conference on Software Architecture. pp. 260–269. WICSA '11, IEEE Computer Society, Washington, DC, USA (2011)
[19] Hofmeister, C., Nord, R., Soni, D.: Applied Software Architecture. Addison-Wesley (2000)
[20] ISO: ISO/IEC CD1 42010, Systems and software engineering — Architecture description (Jan 2010)
[21] Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture. pp. 109–120. WICSA '05, IEEE Computer Society, Washington, DC, USA (2005)
[22] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. Software Engineering, IEEE Transactions on 28(8), 721–734 (Aug 2002)
[23] Kruchten, P., Lago, P., van Vliet, H.: Building up and reasoning about architectural knowledge. In: Hofmeister, C., Crnkovic, I., Reussner, R. (eds.) Quality of Software Architectures, Lecture Notes in Computer Science, vol. 4214, pp. 43–58. Springer Berlin / Heidelberg (2006)
[24] Malaiya, Y.K., Denton, J.: Module size distribution and defect density. In: Proceedings of the 11th International Symposium on Software Reliability Engineering. pp. 62–. ISSRE '00, IEEE Computer Society, Washington, DC, USA (2000)
[25] Mann, H.B., R., W.D.: On a test of whether one of two random variables is stochastically larger than the other. Annals of Mathematical Statistics 18(1), 50–60 (1947)
[26] Miller, J.A., Ferrari, R., Madhavji, N.H.: An exploratory study of architectural effects on requirements decisions. J. Syst. Softw. 83(12), 2441–2455 (Dec 2010)
[27] Mohagheghi, P., Conradi, R., Killi, O.M., Schwarz, H.: An empirical study of software reuse vs. defect-density and stability. In: Proceedings of the 26th International Conference on Software Engineering. pp. 282–292. ICSE '04, IEEE Computer Society, Washington, DC, USA (2004)
[28] Otero, M.C., Dolado, J.J.: Evaluation of the comprehension of the dynamic modeling in uml. Information and Software Technology 46(1), 35 – 53 (2004)
[29] Perry, D.E., Wolf, A.L.: Foundations for the study of software architecture. SIGSOFT Softw. Eng. Notes 17(4), 40–52 (Oct 1992)
[30] Purchase, H.C., Colpoys, L., McGill, M., Carrington, D., Britton, C.: Uml class diagram syntax: an empirical study of comprehension. In: Proceedings of the 2001 Asia-Pacific symposium on Information visualisation - Volume 9. pp. 113–120. APVis '01, Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2001)
[31] Rozanski, N., Woods, E.: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional (2005)
[32] Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika 3(52) (1965)
[33] Stevens, S.: On the theory of scales of measurement. Science 103(2684), 677–680 (1946)
[34] Tyree, J., Akerman, A.: Architecture decisions: Demystifying architecture. IEEE Software 22, 19–27 (2005)
[35] Umphress, D.A., Hendrix, T.D., II, J.H.C., Maghsoodloo, S.: Software visualizations for improving and measuring the comprehensibility of source code. Science of Computer Programming 60(2), 121 – 133 (2006)
[36] Wilk, M.B., Gnanadesikan, R.: Probability plotting methods for the analysis of data. Biometrika 55(1), 1–17 (Mar 1968)
[37] Wohlin, C.: Experimentation in Software Engineering: An Introduction: An Introduction. Kluwer Academic (2000)
[38] Zimmermann, O., Koehler, J., Leymann, F., Polley, R., Schuster, N.: Managing architectural decision models with dependency relations, integrity constraints, and production rules. Journal of Systems and Software 82(8), 1249–1267 (2009)