The Impact of Hierarchies on the Architecture-level Software Understandability - A Controlled Experiment

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

Muhammad Atif Javed
Software Architecture Research Group
University of Vienna
Vienna, Austria
muhammad.atif.javed@univie.ac.at

Uwe Zdun
Software Architecture Research Group
University of Vienna
Vienna, Austria
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. They play a crucial role in the whole development process and in achieving the desired software qualities. This paper presents an empirical study that examines the impact of hierarchies on the architecture-level software understandability. In particular we have studied three different architectural representations of a large-size software system, one with a hierarchical representation where architectural components at all abstraction levels in the hierarchy are shown, and two that do not contain hierarchical abstractions but concentrate only on the lowest level or on the highest level components in the hierarchy. We conducted a controlled experiment in which participants of three groups received one of the three architecture documentations plus the source code of the system and had to answer understandability related questions. Our results show that using the hierarchical architecture leads to: 1) higher quantity of correctly retrieved elements, 2) lower quantity of incorrectly retrieved elements, and 3) higher overall quality of retrieved elements. The obtained results provide empirical evidence that hierarchies play an important role in the context of architectural component models from the viewpoint of the architecture-level software understandability.**

***Keywords*-software architecture; hierarchies; understandability; controlled experiment**

I. Introduction

The main idea of software architecture is to concentrate on the "big picture" of a software system and to enable architects to abstract away the fine-grained details of the implementation and other development artefacts [17]. Software architecture deals with a set of design decisions which, if made incorrectly, may cause a project to be cancelled [16]. Those design decisions represent system's relevant concerns that encompass every aspect of the system under development including system structural decisions, decisions related to the system behaviour, decisions related to the non-functional properties and other decisions related to the business requirements [16]. According to Garlan [7], software architecture plays an important role in at least six aspects of software development: understanding, reuse, construction, evolution, analysis and management.

Architectural component and connector models (or component models for short) are frequently used as a central view of the architectural descriptions of software systems [3]. Note that, according to the software architecture community, an architectural description can comprise multiple views that describe the system concentrating on one of many system aspects, such as logical, implementation, deployment, or process views, and from the viewpoint of different stakeholders, such as end-users, developers, project managers, and business analysts [12], [3]. We concentrate on the architectural information captured in the component models from the perspective of developers and with regard to their relations to the system implementation (i.e., the implementation or development view). From the perspective of the system implementation, component models deal with the coarse-grained components that represent the units of run-time computation or data-storage, and the connectors that are the interaction mechanisms between them [20]. We focus on this view because it is often used to get an initial understanding of the architecture and provides a link for other views, such as the "logical" or "module" views. Since a component in an architectural component model represents a high-level abstraction of the entities in the source code of the software system, it can be broken down into (i.e., is refined by) more fine-grained, technical components or classes that realize the component in the technical design or implementation of the system.

Understandability of a software system is a crucial aspect of the software development process [19]. The difficulty of understanding a software system limits its reuse and maintenance and therefore can influence cost or reliability of software evolution. In the context of architectural component views, understandability is a critical aspect, as one of the main purposes of software architecture is to " ... enable designers to abstract away fine-grained details that obscure understanding and focus on the "big picture:" system structure, the interactions between components, ..." [17]. This, however, is not possible if the given views themselves and/or the links to other design and code artefacts are hard to understand.

Architectural design description of the system can be decomposed into a hierarchy of components that model the system's relevant concerns at different abstraction levels. For instance, according to the guides for software architecture definition in the series of guides for software engineering produced by the Board for Software Standardisation and Control (BSSC) of the European Space Agency [15], architectural decomposition should be constructed as a hierarchical representation where each level models the system's relevant concerns at different levels of abstraction (it starts from a set of high-level components that model high level concerns and results in a set of low-level components that

in combination model the high-level concerns) wherein the decomposition should reach a sufficient level of detail, i.e. provides all the system's relevant concerns. To the best of our knowledge there exist no empirical studies on the impact of hierarchies on the understandability of architectural component models. Existing empirical research into hierarchical models and their impact on understandability mainly targets the domain of conceptual process modelling (see [32] for an overview) such as business process models, ER models, or UML state chart diagrams, and declarative process modelling [33]. While hierarchical structures are recognized as an important factor that influence model understandability, it is not entirely clear whether and when hierarchical structures are beneficial for model understandability [32], [33].

In this study we investigate the impact of hierarchies on the architecture-level software understandability. In particular we have studied three different architectural representations of a large-size software system, one with a hierarchical representation where architectural components at all abstraction levels in the hierarchy are shown, and two that do not use hierarchical abstraction and concentrate either on the lowest level or on the highest level components in the hierarchy. The subjects of the study were 75 students of the Software Architecture lecture at the University of Vienna. They were divided into 3 groups, and each of them studied one of the 3 architectural representations. Our results show that the hierarchical architecture leads to: 1) higher quantity of correctly retrieved elements, 2) lower quantity of incorrectly retrieved elements, and 3) higher overall quality of retrieved elements with regard to understandability related questions, compared to the other 2 architectures. This provides empirical evidence that hierarchies play an important role in the context of architectural component models from the viewpoint of the architecture-level software understandability.

This study is organized as follows: In Section II, we briefly discuss the related work. Section III provides a short overview of the relevant factors for assessing the impact of hierarchies on model understandability and how they are related to component models. In Section IV we describe the study design. Section V describes the statistical methods we applied to analyse our data. Section VI discusses our empirical findings and other related challenges in more detail. In Section VII we discuss the threats to validity of the study. Finally, in Section VIII we conclude and discuss future directions of our research.

II. Related Work

As already mentioned above current empirical research on the impact of hierarchies on model understandability mainly focus on the domains of conceptual and declarative process modelling [32], [33]. In [32] the authors defined a framework for assessing the impact of hierarchies on model understandability by studying the existing research on hierarchically structured conceptual models from the perspective of cognitive psychology. An overview of the existing research on hierarchical models is also provided. In [33] the authors refined a bit a cognitive-psychology-based framework defined in [32] that allows to assess the impact of hierarchies on the understandability of a declarative process model. Moreover the authors discussed the semantics and the application of hierarchies and showed how sub-processes enhance the expressiveness of declarative modelling languages.

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 software systems [8], while another one examines the relationships between some package-level metrics and package understandability [5]. In our previous studies [27], [28] we examined the relationships between the effort required to understand an architectural component, and a number of component level metrics. Several significant correlations and well-fitting prediction models are obtained. In another our study [26] we showed that tangling several system's relevant concerns into one component or scattering them into several components hinders locating those concerns in the system implementation and significantly decreases the understandability.

A lot of studies studied the understandability of different UML models. Some of them studied the layout or visualization aspects of UML models. Purchase et al. [21] revealed that certain visualizations are better than the other depending on the kind of comprehension tasks. Other studies on UML model comprehensibility compare the effect of using different UML diagram types (e.g., sequence and collaboration diagrams). For example, Otero and Dolado took different UML diagrams types, namely sequence, collaboration, and state diagrams, and evaluated the semantic comprehension of the diagrams when used for different application domains [18]. None of these and similar studies examine the impact of hierarchies on the understandability of architectural component models.

As already mentioned above the component models should appropriately encompass all system's relevant concerns and therefore enable conveying a "big picture" of the system. In that way they support the understandability of the whole system and help in creating relationships or mappings between the low level code design and the application domain of the system. Many authors emphasised a mapping between program code and the domain problem (i.e. business related high-level functions) as a key factor for understandability [2], [1]. In the sense of mapping between low-level code design and high-level concerns many authors discussed the concept of feature location as instrumental for the understandability purpose [9]. Capturing system's relevant concerns can be mapped to the system's features modelling in the architecture. A feature is realized functional requirement in the system, and generally also subsumes non functional requirements. Periklis et al. [25] introduced the Feature-Architecture Mapping (FArM) method for feature-oriented development of software product lines. During the analysis of the system and its domain all relevant functional and quality features are extracted. By iteratively refining the initial feature model (FM), a FM is constructed, containing exclusively functional features, whose business logic can be implemented into architectural components (see [25] for more details). In this context the hierarchical organization of the feature model is highlighted as important and necessary.

III. The Impact of Hierarchies on Model Understandability

In this section we provide a short overview of the factors that are relevant for assessing the impact of hierarchies on model understandability and discuss them in relation to component models. Those factors are systematically extracted from the existing empirical research on conceptual and declarative process modelling (see [32], [33] for more details).

With respect to a general-purpose problem solving process two main factors are identified to presumably have an impact on understandability: abstraction and split-attention effect. Abstraction enables grouping of a part of a model into a sub-model that characterizes the group. In that way it reduces the number of elements that have to be considered simultaneously, i.e. it can hide irrelevant information and increase understandability [32]. Besides improving attention management, abstraction presumably supports the identification of higher level patterns [33]. By abstracting and aggregating, information can be easier perceived and an overall model is easier to grasp. However, the sub-models can also have their downsides. When the information from the sub-models needs to be extracted, the reader has to take into account several sub-models, thereby switching attention between sub-models. In addition, the reader has to mentally integrate the sub-process into the parent model, i.e., interpret constraints in the context of the parent process [33]. This effect is called split-attention effect, and it leads to increased mental effort and decreases understandability. For both effects the authors who systematically extracted them assumed that sub-models are presented in a separate window on a computer or printed on a single sheet of paper.

In Section II we emphasised the role of component models for the understandability of a software system as their main purpose is to provide a mapping between the low-level code design and the high-level concerns of the system that many authors indicated as a key factor for understandability. From that perspective the previously mentioned abstraction effect seems to have a significant impact on understandability of component models. Namely, appropriate hierarchical grouping of the system's relevant concerns at different abstraction levels should facilitate the location of those concerns by guiding the attention of the reader to certain parts in the model. After finding relevant parts in the model, the traceability links that link each component to the source code classes that realize the components in the system implementation can further guide the reader to the location of relevant concerns in the system implementation. Furthermore, the abstraction effect supports patterns recognition as mentioned above which strengthens its role in the understanding.

With regard to the influence of the split-attention effect on architectural understandability of component models in comparison to the understandability of conceptual and declarative process models we can say the following. In conceptual and declarative process models a reader mostly faces the problem of checking whether execution traces are supported in the model (i.e. understanding the control flow between different activities). The factors that mostly affect the understandability of those models deal with the execution order, exclusiveness, and concurrent and repeatable execution of activities [23]. In order to understand a given execution trace a reader has to pass over several model parts including the corresponding sub-models and to integrate the implicit constraints that exist on the execution order, exclusiveness, concurrency, and repeatability in the execution. Therefore a reader has to switch attention several times (following the given execution trace) between the sub-models which is further aggravated by the integration of the execution constraints, leading to the split-attention effect. Component models have a static nature, i.e. they focus on a static structure of the system captured by components and connectors that represent the interaction mechanisms between them. Connectors capture the dependencies between components mostly described as a set of required and provided interfaces or services [3]. Therefore tracking the relationships between components in component models seems to be much easier then tracking the relationships between the activities in the process models since we do not have to consider several implicit execution constraints (i.e. execution order, exclusiveness, concurrency, and repeatability) that exist between the process models activities. We simply have to consider the direct relationships between the components described by the required and provided interfaces or services. Having this in mind we can say that the split-attention effect seems to presumably have much less impact on the understandability of component models than on the understandability of declarative and conceptual process models.

Regarding the model size, in order for hierarchies to be beneficial for model understandability, the model must be large enough to benefit from the abstraction [32]. However it is still not clear from the existing empirical research where the threshold related to the size lies. Besides the size of the model, the reader's experience also plays an important role [32]. Experimental settings should be adjusted so that most mental effort is used for problem solving instead of learning, i.e. becoming familiar with the syntax and semantics used for the hierarchical description of the model.

IV. Empirical study description

For the study design we have followed the experimental process guidelines proposed by Kitchenham et al. [10] 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.

A. Goal, hypotheses, and variables

As mentioned above this study examines how the architecture-level software understandability is affected by a hierarchical representation of the system's architecture compared to the architectures where hierarchies are not used. Namely, we compared the understandability for three different architectures of the studied system: one where all the system's relevant concerns captured by the components are represented in the architecture by forming a hierarchical structure with different abstraction levels, one that concentrates only on the highest level components in the hierarchy representing the highest level concerns in the system, and one that concentrates only on the lowest level components

in the hierarchy representing the lowest level concerns in the system. With respect to the discussions provided in Section III we expect that the abstraction effect (that has positive impact on understandability) dominates the split-attention effect (that has negative effect on understandability) in the process of the architectural-level software understandability. Therefore we expect that the hierarchical component model significantly improves architectural understandability in comparison to the other two component models. The hierarchical component model was presented on one page without separately describing sub-components (in a separate window or a piece of paper). This probably led to further decreasing the possible impact of the split-attention effect (see Section III for a detailed discussion about the given effect). To evaluate our results on the understandability related questions we estimated the quantity and quality of retrieved elements using information retrieval measures [13] (see below for more details). The study goal led to the following hypotheses:

H_1: The hierarchical architecture that groups all system's relevant concerns into component hierarchies leads to a higher quantity of correctly retrieved elements compared to the architectures that do not use hierarchical abstractions and provide only the lowest or highest level components in the hierarchy.

H_2: The hierarchical architecture that groups all system's relevant concerns into component hierarchies leads to lower quantity of incorrectly retrieved elements compared to the architectures that do not use hierarchical abstractions and provide only the lowest or highest level components in the hierarchy.

H_3: The hierarchical architecture that groups all system's relevant concerns into component hierarchies leads to higher overall quality of retrieved elements compared to the architectures that do not use hierarchical abstractions and provide only the lowest or highest level components in the hierarchy.

We differentiate 3 dependent and 5 independent variables in our study. The dependent variables include: quantity of correctly retrieved elements, quantity of incorrectly retrieved elements, and overall quality of retrieved elements. The dependent variables are accessed by using recall, precision, and F-measure, respectively, the standard metrics used to evaluate the performances of information retrieval systems [13]. The level of scaling for information retrieval measures falls in the interval [0,1]. Each question about the studied system requires a set of system elements (source code classes, packages, and/or architectural components) as an answer. All the questions in the study are subjective, open-ended questions. Because answers to the questions consist of a list of system elements, the following aspects are taken into consideration to calculate the information retrieval statistics:

- The set of correct elements expected in the solution to question i (C_i).
- The set of elements mentioned in the solution to question i by participant p ($M_{p,i}$).

Based on the above definition, recall and precision are computed for every subjective question. Recall is the percentage of correct matches retrieved by a study subject, while precision is the percentage of retrieved matches that are actually correct.

$$Recall_{p,i} = \frac{\mid M_{p,i} \cap C_i \mid}{C_i} \qquad Precision_{p,i} = \frac{\mid M_{p,i} \cap C_i \mid}{M_{p,i}}$$

Because recall and precision measure two different concepts, it can be difficult to balance between them. We used F-measure, a standard combination of recall and precision, defined as their harmonic mean, to measure the global correctness of answers from the study participants.

$$F - measure_{p,i} = 2 * \frac{precision_{p,i} * recall_{p,i}}{precision_{p,i} + recall_{p,i}}$$

The independent variables used in our study concern the participants experience (programming experience, commercial programming experience, and experience in programming Java applications), group affiliation (3 different groups of participants) and time spent in the study. With respect to the goal of our study 3 different treatments are defined for the participants. Each treatment (group of participants) is explicitly told to answer the questions aimed at gaining the architecture-level understanding of a representative subject system, and each group is provided with a different architecture of the studied system. The independent variables could have an influence on the dependent variable, which is eliminated by balancing the characteristics between the given 3 groups of participants.

Description	Scale type	Unit	Range
Quantity of correctly retrieved elements	Interval	Points	[0,1]
Quantity of incorrectly retrieved elements	Interval	Points	[0,1]
Overall quality of retrieved elements	Interval	Points	[0,1]

Table I
DEPENDENT VARIABLES

The dependent variable together with its scale type, unit, and range is shown in Table I. The independent variables are shown in Table II. The range for the variable "Group affiliation" is the following: "Group A_{low}" corresponds to the participants who have studied the architecture with the lowest-level components in the hierarchy, "Group $A_{hierarchy}$" corresponds to the participants who have studied the hierarchical architecture where the components are appropriately organized into different abstraction levels, and "Group A_{high}" corresponds to the participants who have studied the architecture with the highest-level components in the hierarchy.

B. Study Design

The execution of the study used to test the hypotheses took place as a part of the Software Architecture lecture at the University of Vienna, Austria, in the Summer Semester 2014.

1) Subjects: The subjects of the study were 75 students of the Software Architecture lecture at the University of Vienna.

Description	Scale type	Unit	Range
Programming experience	Ordinal	Years	4 categories: 0, [1-3), [3-7), >=7
Commercial programming experience	Ordinal	Years	4 categories: 0, [1-3), [3-7), >=7
Experience in programming Java applications	Ordinal	Years	4 categories: 0, [1-3), [3-7), >=7
Time	Ordinal	Minutes	90 minutes (max)
Group affiliation	Nominal	N/A	Group A_{low}, Group $A_{hierarchy}$, Group A_{high}

Table II
INDEPENDENT VARIABLES

2) Objects: The software system to be studied by participants was WebWork [14], version 2.2, an open source Java-based web application framework. It is built to improve developer productivity and simplify code. It provides robust support for building reusable UI templates, such as form controls, UI themes, dynamic form parameter mapping to Java Beans, robust client and server side validation, etc. The system has around 322K SLOC and can be considered as relatively large[1]. As we already mentioned in Section III, in order that hierarchies are beneficial for model understandability, the model must be large enough to benefit from the abstraction. Therefore we selected a large-size system to be studied. The choice of using this particular system is further motivated by the following factors:

- It is an open source system, which enables us to conduct the study and disseminate its results.
- It utilizes several design patterns and best practices which is also in line with the course lectures and with which the participants were sufficiently familiar.
- It utilizes elegant solutions to overcome the limitations observed in other web frameworks and better understand a domain problem and a logic behind it.

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

Architectural documentation about the WebWork system version 2.2: The system's documentation describes the conceptual architecture and lists technologies and frameworks used in the implementation. Besides textual description, a UML component diagram is used to illustrate the components in the system, and their inter-relationships. Participants were also provided with a set of traceability links, showing the relations between architectural components and their realized source code classes. Those links help in locating the system's relevant concerns, captured by the components, in the system implementation, i.e. source code.

Each of the 3 groups of participants received one architectural representation of the system together with the corresponding traceability links. The first, hierarchical architecture, appropriately groups all system's relevant concerns captured by the components into hierarchy with properly assigned abstraction levels. This means that first the lowest level components that should capture all system's relevant concerns are identified. The source code classes that implement the given concerns in the system implementation are assigned to the corresponding components in which way the traceability links are formed. The lowest level components are further grouped into higher level components that represent higher level concerns and the process is repeated till no more further grouping is possible. This process requires human expertise and some guidelines that support the process can be found in the corresponding literature on the software architecture definition guides [15] and the system's feature location and modelling in the architecture [9], [25], already mentioned in Sections I and II. The hierarchical architecture and the corresponding traceability links are created by two experienced software architects who deeply studied the given system and its documentation. The other two architectures are easily derived from the hierarchical architecture.

The hierarchical architecture of the WebWork system consists of 41 components at the lowest abstraction level. At the highest level of abstraction 5 composite components are modeled containing 37 of those 41 low-level components. That is, 4 low-level components are not part of any higher-level component[2]. The hierarchical architecture provides further structure through 3 composite subcomponents (InversionOfControl, PropertyInterceptors, and WorkflowInterceptors) which are part of two highest level components. That is, the hierarchical architecture consists of 3 levels. Figure 1 shows the given hierarchy of components without relationships between them as well as the detailed architecture of Component ActionDispatching. As it can be seen from the figure relationships between the components in the architectures are established using the required and provided interfaces as well as the assembly and delegation connectors (typical notation used in the UML component diagrams). The components are organized in an appropriate layout so that the relationships between them can be easily grasped. With respect to what we mentioned above the highest-level architecture consists of the following 9 components UserInterfaceTags, Views, ActionDispatching, PortletApplications, Helpers, XWorkMainClasses, Configuration, Interceptors, and ExternalIntegrations, and the relationships among them where none of the subcomponents of any composite component is shown. The lowest-level architecture consists of all 41 components that do not have any subcomponent and the relationships among them (none of the composite components is shown). A part of the lowest-level architecture with some components and their relationships is shown in Figure 2.

Browser-based source code access: Browser-based access to the source code of the system was provided in a Lab environment on prepared computers. All source code classes were grouped into the corresponding components so that the participants can easily relate the components in the system to their realized source code classes.

A questionnaire to be filled-in by the participants during the study execution: On the first page of the questionnaire, the participants had to rate their experience, i.e. programming experience, commercial programming experience, and Java programming experience. The subsequent pages contain the understanding questions. In the context

[1]ISBSG Repositories (The International Software Benchmarking Standards Group), 2007.

[2]Those components were included in the architecture that concentrates on the highest level components in order to provide a complete set of components in the system. That is, the highest-level architecture consists of 5 composite components + 4 components = 9 components.

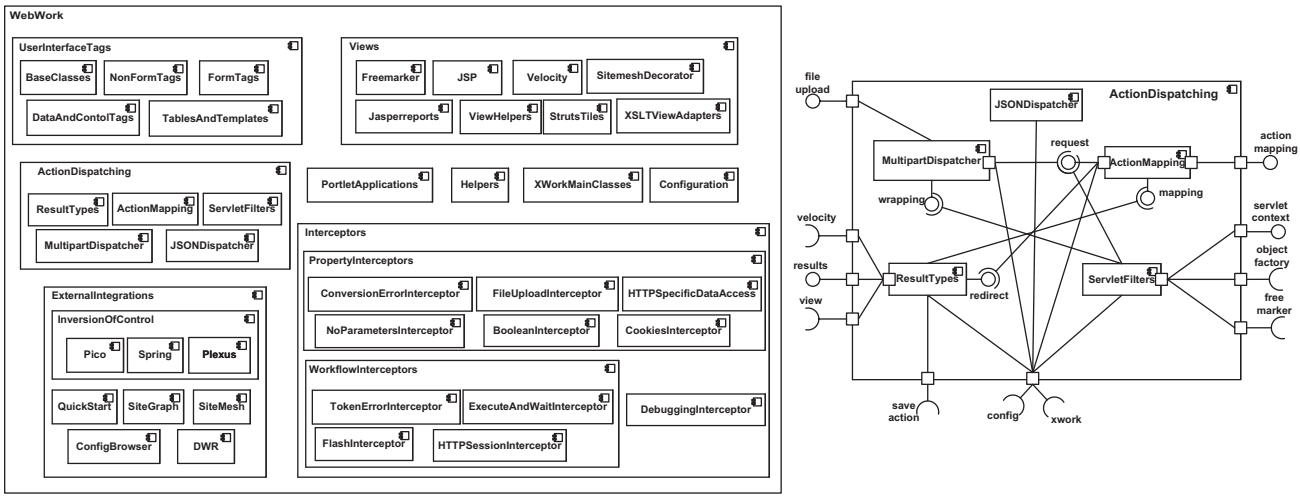

Figure 1. WebWork Components Hierarchy and the Detailed Architecture of Component ActionDispatching

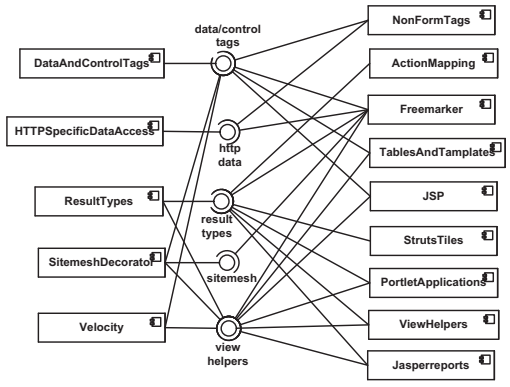

Figure 2. A Part of the Lowest-level Architecture

ID	Description	Comprehension activities
Q1	Investigating parts of the system related to automatic adjustments of different attributes and properties on the actions.	A1, A9, A3
Q2	Investigating parts of the system related to displaying of a web page (user interface) without those that manage the execution flow and data access.	A1, A9, A3
Q3	Investigating the Model, the View, and the Controller part of the MVC pattern used in the system.	A7, A1, A8
Q4	Investigating parts of the system integrated from outside of the WebWork that communicate with the *XWork* but not with the miscellaneous helper classes.	A4, A5, A6
Q5	Investigating the impact of changes in the classes that monitor the resource management activities.	A2, A8, A7
Q6	Investigating the common data flow during the life cycle of a Web Work request at runtime.	A4, A5, A9
Q7	Investigating parts of the system related to the mapping between the HTTP requests and the corresponding actions as well as parts that are directly dependent on them.	A3, A4, A6

Table III
QUESTIONNAIRE FOR THE ARCHITECTURE-LEVEL SOFTWARE UNDERSTANDING

of the questions, two important criteria are applied: (i) the questions should be representative for key understanding contexts, and (ii) they should be imaginatively constructed to measure the deeper understanding of the participant groups. With regard to this, nine principal understanding activities that are typically performed during real-world software understanding are applied. Those activities are defined in the work by Pacione et al. [19] (please refer to [19] for a detailed description). Guided by these activities, 7 representative questions[3] (shown in Table III) are defined that highlight important aspects of the WebWork system at both a high-level of abstraction (architecture-level) and a low-level of abstraction (source-code-level). The last column in the table shows the mapping between the questions and the aforementioned nine principal understanding activities.

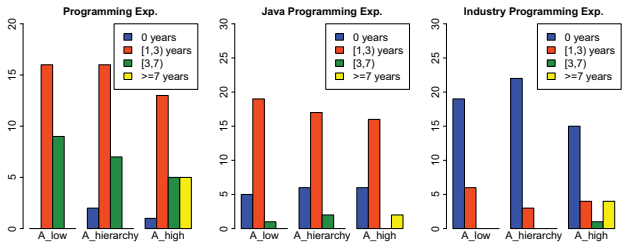

Figure 3. Experience of the Participants

C. Execution

1) Preparation: As explained in Section IV-B, the study was conducted in the practical part of the Software Architecture course at the University of Vienna, Austria. The total time limit for the whole study was 2 hours. The participants were randomly assigned to the three groups. From Figure 3 we can see that the experience of the participants in all three groups is quite well balanced. We also confirmed it statistically by pursuing the Cliff's test and concluded that there is no significant difference in the experience between the 3 groups (see below for details about the given test).

[3]The number of questions is estimated in a pre-study, conducted with our colleagues, to ensure that the participants have fairly enough time to study all of them.

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 3.

According to the experience of the participants we can say that the participants have medium to high programming experience (most of them have [1,3) and [3,7) years of programming experience while some of them have more than 7 years of experience), and medium Java programming experience. Only a very few participants have industrial programming experience. The participants who have 0 years of programming experience are excluded from the consideration for the statistical analysis pursued in Section V. Additionally, two participants from the first group ("Group A_{low}") and four participants from the third group ("Group A_{high}") answered just few questions and they were also excluded from the analysis because this would just introduce bias in the results.

Variable	Group affiliation	Participants	Mean	Median	Std. Dev.
Recall	Group A_low	23	0.4584	0.4673	0.1743
	Group A_hierarchy	23	0.6707	0.6610	0.1592
	Group A_high	20	0.4204	0.3917	0.1676
Precision	Group A_low	23	0.3488	0.3374	0.1453
	Group A_hierarchy	23	0.6309	0.6131	0.1416
	Group A_high	20	0.4289	0.4345	0.1570
F-measure	Group A_low	23	0.3658	0.3703	0.1525
	Group A_hierarchy	23	0.6316	0.6093	0.1458
	Group A_high	20	0.3865	0.3644	0.1573

Table IV
RECALL, PRECISION, AND F-MEASURE – DESCRIPTIVE STATISTICS

Table 4 shows the means of the recall, precision, and F-measure values per question for each group. We see that "Group $A_{hierarchy}$" has higher scores for most of the questions. However the other 2 groups scored quite well for most of the questions. For the questions Q3 and Q6 almost the same scores can be observed. For the question Q3 related to the investigation of the MVC pattern used in the system, the participants probably utilized well the package structure of the system implementation that can help in recognizing some high level concerns. When they located some of the potential classes related to each part of the pattern they presumably followed the relationships between the classes to search for the classes with similar functionality. Regarding the question Q6 that deals with the common data flow during the life cycle of the WebWork request at runtime, the hierarchical architecture did not provide significant benefit since the participants had to locate the low level source code elements based on their behaviour. Regarding the question Q7 groups "Group $A_{hierarchy}$" and "Group A_{low}" have similar scores while group "Group A_{high}" has lower score compared to them. The reason for that might be that the first 2 groups both utilized well the architecture since the required component, that closely determines the location of the required concern in the source code, is presented in both architectures (it is the lowest level component in the system). The only question where the participants from groups that used non hierarchical architectures scored poorly is the question Q5. It seems that for those groups it was really hard to locate the entities that monitor the resource management (see Q5 in Table III) compared to the group that used the hierarchical architecture were the participants presumably recognized the components responsible for that (like for example the "Inversion of Control" component).

V. ANALYSIS

A. Descriptive Statistics

The mean, the median, and the standard deviation of the recall, precision, and F-measure values for all three groups of participants are shown in Table IV. The statistics in the figure is calculated from the "per participant" results that describe the scores of each participant. In total, the mean and median values in "Group $A_{hierarchy}$" are higher than those in the other two groups ("Group A_{low}" and "Group A_{high}").

B. Testing Hypotheses

Based on the data obtained from the questionnaire we applied the following statistical analyses with our data using the programming language R [22]:

- The Shapiro-Wilk test [24] for testing the data normality
- The Cliff's method in conjunction with the Hochberg's method [29] for comparison of a location shift between more than two variables

Parametric statistical tests are generally more powerful than the analogous non-parametric tests. In order to apply parametric tests certain assumptions must be true: data normally distributed, homogeneity of variance through the data, at least an interval level of the data, and independence of scores in the response variable(s) (i.e., what you get from one subject should be in no way influenced by what you get from any of the others) [6].

As the first step, we tested the normality of the data by applying the Shapiro-Wilk normality test in R. As obtained p-values are lower that 0.05 which means that our data show significant variation from the normal distribution. Therefore we decided to apply non-parametric tests to test our hypotheses. The classic methods is the Kruskal Wallis test used to examine if there is a significant difference in the location shift between the 3 groups in the study with respect to the observed measures (recall, precision, and F-measure). However, we use the more robust Cliff's method in conjunction with the Hochberg's method (to control the probability of one or more type I error) that allows heteroscedasticity (different variances in the tested groups) and performs well when tied values can occur [29].

The results of the Cliff's method for all pairs of the 3 groups are shown in Figure V. Particularly, the p-values that show if there is a significant difference between the groups, the corresponding critical p-values, and the p-hat values that measure the effect sizes are shown. If the p-values are lower than the corresponding critical p-values it means that there exists a significant difference between the groups [29]. The effect size indicates how strong is the obtained difference between the groups. Values around 0.556 indicate small effect size, around 0.638 medium effect size, and around 0.714 large effect size [11]. From the obtained results we see that for all observed measures (recall, precision, and F-measure) there exists a significant difference in the location

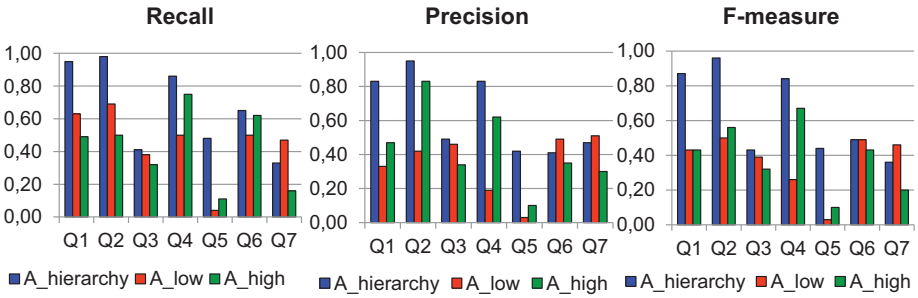

Figure 4. The Means of the Recall, Precision, and F-measure Values for Each Question

shift between "Group $A_{hierarchy}$" and "Group A_{low}" as well as between "Group $A_{hierarchy}$" and "Group A_{high}", while the difference between "Group A_{low}" and "Group A_{high}" is not significant (the corresponding p-values are greater than their critical p-values). All calculated effect sizes can be assumed as large and indicate strong difference in the location shift between the groups. Table IV indicates the higher values for the observed measures for the group that used the hierarchical architecture ("Group $A_{hierarchy}$") compared to the other 2 groups in the study. Given the results from the analysis undertaken, it has been demonstrated that all 3 hypotheses of our study are supported, i.e. the hierarchical architecture that groups all system's relevant concerns into a component hierarchy leads to: 1) higher quantity of correctly retrieved elements, 2) lower quantity of incorrectly retrieved elements, and 3) higher overall quality of retrieved elements, on the understandability related questions compared to the architectures that do not use hierarchical abstractions and show only the lowest level or the highest level components in the hierarchy.

Recall (PV:p-val, PC:p-crit, PH:p-hat)			Precision (PV:p-val, PC:p-crit, PH:p-hat)			F-measure (PV:p-val, PC:p-crit, PH:p-hat)		
	A_low	A_hier		A_low	A_hier		A_low	A_hier
A_hier	**PV=1e-04 PC=0.025 PH=0.822**	--	A_hier	**PV=1e-04 PC=0.017 PH=0.923**	--	A_hier	**PV=1e-04 PC=0.025 PH=0.902**	--
A_high	PV=0.46 PC=0.05 PH=0.43	**PV=1e-04 PC=0.017 PH=0.856**	A_high	PV=0.16 PC=0.05 PH=0.63	**PV=2e-04 PC=0.025 PH=0.832**	A_high	PV=0.78 PC=0.05 PH=0.53	**PV=1e-04 PC=0.017 PH=0.867**

Table V
THE RESULTS OF THE CLIFF'S METHOD

VI. DISCUSSION

In the view of the obtained results we provide empirical evidence that hierarchies play an important role in the context of architectural component models from the viewpoint of the architecture-level software understandability. More concretely we studied a system whose architecture comprises 41 lowest level components in the hierarchy (1st abstraction level), 3 composite sub-components (2nd abstraction level), and 5 highest level composite components (3rd abstraction level). We showed that two of the architectures that do not use the hierarchical structure of components (i.e. they concentrate either on the lowest or highest level components in the hierarchy) lead to lower understandability scores compared to the hierarchical architecture. The non-captured components apparently hampered finding the location of the system's relevant concerns in the architecture. Consequently it hampered also finding the location of those concerns in the system implementation.

Generally speaking it is quite reasonable to expect that if the number of non-captured components in the system's architecture increases, the understandability of the system decreases. However examining precisely how the understandability would be affected when different numbers of components from different abstraction levels are not captured in the architecture remains to be a big challenge that requires several studies and much more resources, i.e. participants, time, etc. Another challenge is to find the threshold for the system size starting from which hierarchies are beneficial. This challenge is also unresolved for other kinds of models studied in the existing empirical research mentioned in Section III.

Regarding the two main factors identified to be relevant for assessing the impact of hierarchies on model understandability (see Section III) we showed that the abstraction effect plays a very important role for the understandability of component models which is quite expected since one of their main purposes is to " ... enable designers to abstract away fine-grained details that obscure understanding and focus on the "big picture:" system structure, the interactions between components, ..." [17]. Regarding the split-attention effect the case where many sub-components in the system exist and each is presented in a separate window or on a separate piece of paper could be further examined. However, in the light of the discussion provided in Section III, this effect seems to presumably have a much less significant impact on component models than on process models.

VII. VALIDITY EVALUATION

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

1) 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 (23, 23, and 20 students in "Group A_{low}", "Group $A_{hierarchy}$", and "Group A_{high}", respectively). In a between subjects-design, 20 participants are recommended to detect a large effect in the one way ANOVA test with a power of 0.8 and a significance level of 0.05 [4]. In the corresponding

non-parametric test maximum 15 % more participants can be expected [31] leading to 23. As we obtained that there is a statistically significant difference between the studied groups (with a large effect size) for the given sample size we would be able to detect even tiny differences between the groups if the sample size increases. Therefore there is a low threat to conclusion validity of our results.

2) Construct validity: The construct validity is the degree to which the independent and the dependent variables are accurately measured by the appropriated instruments. The interpretation of the answers to the questions might result in a threat to validity of the dependent variable because the answers to the questions consist of a list of system elements (e.g., architectural components, source code classes, packages). We mitigated this risk by calculating the standard information retrieval metrics for recovered answers from all questions. We argue that information retrieval measures allow objective evaluation of the correctness of questions rather than intuitive or ad-hoc human measures. We conclude that this potential threat is mitigated to a large degree.

3) 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 deal with the following issues:

Differences among subjects: The variation in human performance might distort the results of the study, and then the performance differences would not arise from the difference in treatments. In this particular study, the participants' experience is quite well balanced among the three groups in the study and there is no significant difference among the groups (see Section IV-C1). Thus, this factor is not seen as a strong threat to validity.

Measuring method: A potential threat to validity might be that the understanding of the questionnaire could have been biased towards "Group $A_{hierarchy}$". Answering some of the questions might be easier for that group because the architecture for that group reduces the decision space by pointing to the component or the set of components related to the examined concern. However, those questions are based on the established comprehension framework related to examining the relevant concerns of (a part of) the system and how those concerns are interrelated [19]. The established task framework also ensures that many aspects of typical understanding contexts are covered. Beside the usage of the common framework the questions are imaginatively constructed to measure the deeper understanding of the groups (see Section IV-B3).

4) 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:

The system that is used: The fact that we used only one object in the study might introduce the risk of generalizing the results. Generalization aspects are discussed in more detail in Section VI.

Subjects: The participants' population in the study might not be sufficiently competent. This might influence the results of the study. In this study, all the participants had knowledge about software development and software architecture (UML modelling), as well as of software traceability. They all studied the previous lectures of at least the software architecture course and have medium to high programming experience (see Figure 3). The participants were also familiar with the technologies, concepts and frameworks used in the study since they were taught about it in the course the study took place. However we are aware that more empirical studies with professionals need to be carried out in order to generalize the results.

VIII. Conclusions

This paper presents a controlled experiment on the impact of hierarchies in component models on the architecture-level software understandability. We have studied 3 different component models of an open source Java-based web application framework called WebWork. One of the 3 component models used a hierarchical representation where architectural components at all abstraction levels in the hierarchy are shown, while the other two do not use hierarchical abstractions and focus only on the lowest or highest level components in the hierarchy.

Our results show that using the hierarchical architecture leads to: 1) higher quantity of correctly retrieved elements, 2) lower quantity of incorrectly retrieved elements, and 3) higher overall quality of retrieved elements with regard to understandability related questions. The obtained results provide empirical evidence that hierarchies play an important role in the context of the architecture-level software understandability. The effect of abstraction plays a very important role since it facilitates the location of the system's relevant concerns by guiding the attention of the reader to certain parts of the architecture and thereafter to the corresponding part in the system implementation. Having in mind that the architecture plays a crucial role in the whole development process, improving our knowledge on creating an understandable architecture helps us to improve the overall quality of the software it represents.

In our future work we plan to study the systems with more abstraction levels and to examine more precisely how the understandability would be affected when different numbers of components from different abstraction levels are not captured in the architecture. Also it would be interesting to study the case where many sub-components in the system exist and each is presented on a separate piece of paper which can potentially cause the so-called split-attention effect.

Acknowledgement

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

References

[1] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster. Program understanding and the concept assignment problem. *Commun. ACM*, 37(5):72–82, May 1994.

[2] R. Brooks. Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, 18(6):543 – 554, 1983.

[3] 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.

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

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

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

[7] D. Garlan. Software architecture: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 91–101, New York, NY, USA, 2000. ACM.

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

[9] H. Kazato, S. Hayashi, T. Kobayashi, T. Oshima, S. Okada, S. Miyata, T. Hoshino, and M. Saekii. Incremental feature location and identification in source code. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 371–374, March 2013.

[10] 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.

[11] H. C. Kraemer and D. J. Kupfer. Size of treatment effects and their importance to clinical research and practice. *Biological Psychiatry*, 59(11):990 – 996, 2006.

[12] P. Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6):42–50, Nov. 1995.

[13] F. Lancaster. *Information Retrieval Systems: Characteristics, Testing, and Evaluation*. Information Sciences Series. Jon Wiley & Sons, 1979.

[14] P. Lightbody and J. Carreira. *WebWork in Action*. In Action Series. Manning, 2006.

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

[16] N. Medvidovic. Moving architectural description from under the technology lamppost. In *Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference on*, pages 2–3, Aug 2006.

[17] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, 14(3):54–62, May 1999.

[18] M. C. Otero and J. J. Dolado. Evaluation of the comprehension of the dynamic modeling in uml. *Information and Software Technology*, 46(1):35–53, 2004.

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

[20] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, Oct. 1992.

[21] H. C. Purchase, L. Colpoys, M. McGill, D. Carrington, and C. Britton. Uml class diagram syntax: An empirical study of comprehension. In *Proceedings of the 2001 Asia-Pacific Symposium on Information Visualisation - Volume 9*, APVis '01, pages 113–120, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

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

[23] H. Reijers and J. Mendling. A study into the factors that influence the understandability of business process models. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(3):449–462, May 2011.

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

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

[26] S. Stevanetic and U. Zdun. Empirical study on the effect of a software architecture representation?s abstraction level on the architecture-level software understanding. In *International Conference on Quality Software 2014 (QSIC)*, October 2014.

[27] S. Stevanetic and U. Zdun. Exploring the relationships between the understandability of architectural components and graph-based component level metrics. In *International Conference on Quality Software 2014*, October 2014.

[28] S. Stevanetic and U. Zdun. Exploring the relationships between the understandability of components in architectural component models and component level metrics. In *18th International Conference on Evaluation and Assessment in Software Engineering (EASE 2014)*, May 2014.

[29] R. Wilcox. Chapter 7 - one-way and higher designs for independent groups. In R. Wilcox, editor, *Introduction to Robust Estimation and Hypothesis Testing (Third Edition)*, Statistical Modeling and Decision Science, pages 291 – 377. Academic Press, Boston, third edition edition, 2012.

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

[31] D. Wolfe. Nonparametrics: Statistical methods based on ranks and its impact on the field of nonparametric statistics. In J. Rojo, editor, *Selected Works of E. L. Lehmann*, Selected Works in Probability and Statistics, pages 1101–1110. Springer US, 2012.

[32] S. Zugal, J. Pinggera, B. Weber, J. Mendling, and H. Reijers. Assessing the impact of hierarchy on model understandability a cognitive perspective. In J. Kienzle, editor, *Models in Software Engineering*, volume 7167 of *Lecture Notes in Computer Science*, pages 123–133. Springer Berlin Heidelberg, 2012.

[33] S. Zugal, P. Soffer, C. Haisjackl, J. Pinggera, M. Reichert, and B. Weber. Investigating expressiveness and understandability of hierarchy in declarative business process models. *Software & Systems Modeling*, June 2014.