Two Controlled Experiments on Model-based Architectural Decision Making

Ioanna Lytra*, Patrick Gaubatz, Uwe Zdun

Software Architecture Research Group, University of Vienna, Austria

Abstract

Context: In recent years, architectural design decisions are becoming more and more common for documenting software architectures. Rather than describing the structure of software systems, architectural decisions capture the design rationale and – often reusable – architectural knowledge. Many approaches and tools have been proposed in the literature to support architectural decision making and documentation (for instance, based on models, ontologies, or templates). In this context, the capturing, organization, and effective reuse of architectural knowledge has gained a lot of attention. **Objective:** However, there is little empirical evidence about the supportive effect of reusable architectural knowledge on the effectiveness and efficiency of architectural decision making. **Method:** To investigate these aspects, we conducted two separate controlled experiments with software architecture students in which we tested the supportive effect of reusable decision models in decision making and documentation. **Results:** Our results show that the use of reusable decision models can significantly increase both the efficiency and the effectiveness of novice architects. **Conclusion:** We can report, that our findings are in line with similar studies and support the claims regarding reusable architectural design decisions in principle.

Keywords: architectural design decision, architectural decision model, architectural knowledge, controlled experiment

1. Introduction

In recent years, architectural design decisions (ADDs) have been promoted to first class citizens in software architecture documentations [1]. Rather than documenting the structure of software systems (e.g., components and connectors), ADDs contribute to the capturing of design rationale. There are numerous attempts on documentation and leveraging of design rationales with focus on

*Corresponding author: Ioanna Lytra, Faculty of Computer Science, University of Vienna, Währingerstraße 29, 1090 Vienna, Austria; Phone, +43-1-4277-78523

Email addresses: ioanna.lytra@univie.ac.at (Ioanna Lytra), patrick.gaubatz@univie.ac.at (Patrick Gaubatz), uwe.zdun@univie.ac.at (Uwe Zdun)

the reduction of architectural knowledge (AK) vaporization [2], reusability of ADDs [3], and AK sharing [4]. Apart from that, the documentation of ADDs for providing architectural guidance in software projects has gained much attention in industrial practice [5, 6], lately. In this context, capturing the design solutions and their rationale is important not only for the experienced software architects but also for novice software designers who need to be educated on the existing AK and the systematic reasoning on ADDs to avoid both reinventing the wheel and making ADDs of bad quality.

Reusing ADDs can contribute to simplifying architecting [7]. Thus, addressing systematic documentation of ADDs and providing guidance during decision making for recurring design issues, the use of reusable ADD models has been proposed in the literature [3]. Similar to leveraging patterns for architectural decision making [2], reusable ADD models provide proven solutions – both application generic and technology specific – to various design issues along with their forces and consequences. Examples of reusable ADD models that have been documented cover solutions for designing service-oriented architectures (SOA) [8] and service-based platform integration solutions [9].

A few reusable ADD models and related tools that support their management (such as [10]) have been evaluated in real-life contexts. For instance, Zimmermann et al.'s ADD model consisting of 300 ADDs from the SOA domain covering various aspects such as Web service integration and transaction management has been used by practitioner communities and in industrial projects [8]. However, no feedback or empirical evidence has been gathered on whether and to which extent reusable ADD models are beneficial (i.e., they support effectiveness and efficiency of architects) in the architectural decision making process. While a few studies have investigated how reusable AK management and sharing is practiced in industrial contexts [5, 11] and have validated the supportive effect of pattern-based reusable AK in the decision making process [12], the software architecture community lacks empirical evidence on the positive impact of reusable AK on ADD making and documentation. Such empirically-grounded findings are important not only for validating the benefits of reusable AK in practice, but also for understanding, improving, and supporting the management and leveraging of reusable ADDs.

Therefore, we conducted two controlled experiments with students to test whether the use of reusable ADD models increases the efficiency and effectiveness of architects in the decision making and documentation process. We explicitly considered software architecture students in our evaluation, as reusable ADD models are supposed to be used as guidance models by trainers for systematically teaching patterns and technology best practices to new or inexperienced members in a software development team [13]. In the two controlled experiments, 49 and 122 students, respectively, with background in software architecture and design patterns, were asked to make and document ADDs while designing the architecture of two different software systems. For this, a Web-

based tool support, called CoCoADvISE[1], was provided to the experiment and control groups. Both the experiment and control group received material with related patterns and technology documentations and could use the tool to make and document decisions. Contrary to the control group, the tool provided additional semi-automated decision making guidance based on reusable ADD models for the participants of the experiment group. We found that participants who were supported by our semi-automated decision making guidance approach ...

- delivered more documented ADDs.
- delivered ADDs of better quality.
- invested less time for documenting ADDs.

The remainder of the paper is structured as follows. We give an overview of the approaches related to architectural decision making and documentation and compare existing architectural decision management tools with CoCoADvISE in Section 2. We present our Web-based tool CoCoADvISE for decision making and documentation and discuss its main concepts in Section 3. In Sections 4 and 5 we describe our experimental settings, as well as the analysis of the results for the two controlled experiments we conducted. Our findings, implications, and validity threats are discussed in Section 6, and finally, Section 7 concludes with a summary of our contributions and discusses future work.

2. Related Work

In this section, we discuss the concept of ADDs, present existing tools and methods for decision making and documentation, and summarize the few empirical studies related to ADDs that exist in the literature.

2.1. Architectural Design Decisions

ADD documentations contain not only the resulting designs but also the justification, the strengths and weaknesses, as well as alternatives for the selected design solutions. Thus, software architects capture ADDs for analyzing and understanding, as well as sharing and communicating the rationale and implications of these decisions. Apart from that, the documentation of ADDs prevents the potential loss of AK, a phenomenon which is known as *architectural knowledge vaporization* [1, 2]. There are numerous attempts on supporting ADDs and capturing their design rationales. Clements et al. suggest a general outline for documenting architectures and guidelines for justifying design decisions [15] while Tyree and Akerman present a rich template for capturing

[1]CoCoADvISE is the web-based version of our previous tool ADvISE for decision making and documentation [14] and shares common concepts with other reusable ADD approaches that have been documented in the literature (such as [3]). CoCoADvISE was developed for the needs of the controlled experiments and in order to provide additionally collaboration support.

and documenting several aspects of ADDs [16]. A different technique proposed by Lee and Kruchten aims at establishing formalized ontological descriptions of architectural decisions and their relationships [17]. Zimmermann et al. use decision meta-models to capture reusable AK [3] to be reused among different projects of the same domain. In addition, patterns are regarded proven knowledge for capturing ADDs and their rationale [2] and are considered often in the aforementioned approaches.

Numerous tools for the management of ADDs have been proposed in the literature [18–20]. In addition, a substantial amount of work has been done in the direction of documenting the AK using architectural decision modeling (refer to [18] for a comparison of existing architectural decision models and tools). For instance, Jansen and Bosch propose a meta-model for capturing decisions that consist of problems, solutions and attributes of the AK [1]. Zimmermann et al.'s meta-model for capturing ADDs [6] consists of three core domain entities: *Architectural Decision (AD)* related to one or more *ADTopics* organized in *ADLevels*, entailing *ADAlternatives*, the selection of which leads to an *ADOutcome*. The advantage of such ADD models is that they are reusable and can be used as guidance for architectural decision making activities, whenever recurring design issues emerge. Reusable ADD models share common concepts with patterns (see [2]), that is, they both provide proven solutions for specific design issues along with their motivation and rationale. The main difference is that reusable ADD models provide the means for defining formally more complex relationships for ADDs (e.g., the selection of a design option may exclude a design solution). Furthermore, they allow us to capture except for generic knowledge – usually addressed by patterns – also domain and technology specific AK. Yet, the relationship between architectural patterns and reusable ADD models can be eventually synergetic [3], for instance, reusable decision models can be integrated with patterns and guide the selection of patterns. Various reusable ADD models have been documented in the literature, covering SOA-related solutions [8], service-based platform integration [9], the design of domain specific languages [21], and model and metadata repositories [22].

In our empirical study, we focus on the evaluation of reusable AK in the form of reusable ADD models. For this, we provide reusable ADD models for the participants of the experiment groups of the two controlled experiments similar to the aforementioned reusable ADD models.

2.2. Tools for Architectural Decision Making and Documentation

Several tools have been developed to ease capturing, managing and sharing of architectural decisions. In most of the cases, the focus is set on the manipulation of architectural decision artifacts and their relationships, and the capturing and reuse of AK, as well as collaboration aspects. In our work, we do not intend to develop "yet another tool" for ADD management but rather to implement existing concepts in architectural decision support such as reusable architectural decision models [3] and the Questions-Options-Criteria (QOC) approach [23] and provide semi-automated tool support integrating these concepts. Our main goal is to gather empirical evidence on the supportive effect of reusable ADDs

in architectural decision making. In this section, we discuss existing tools for architectural decision making and documentation and compare these to CoCoADvISE, the Web-based tool we have evaluated in our empirical study.

PAKME [24] and ADDSS [25] are some of the first attempts to develop tools for decision making and documentation. PAKME aims at providing AK management support for knowledge acquisition, maintenance, retrieval, and presentation, and contains features for capturing and classifying ADDs, integrating pattern-based knowledge, and relating ADDs with quality-based scenarios and requirements. Similar to PAKME, ADDSS can be used for storing, managing and documenting ADDs. CoCoADvISE targets both decision making and documentation. For the decision making, reusable ADD models are leveraged [3] and for the documentations, a text template like the one introduced by Tyree and Akerman [16] is used. Unlike tools that focus on the visualization and understandability of ADDs (such as [26, 27]), we mainly target decision guidance and automation of steps in decision making and documentation.

Many tools in the literature support the integration of ADDs in the software architecture and development processes. For instance, Archium aims primarily at capturing architectural decisions and weaving them into the development process, that is, binding them with models and system implementation [28]. Also, the ADVERT approach targets the capturing of pattern-specific ADDs with their rationale along with decision-relevant requirements, quality attributes, and architectural elements [29]. In our previous work, we have integrated ADDs modeled in CoCoADvISE tool with architectural views, in particular, component-and-connector views [14]. These aspects are out of the scope of this paper though.

Some of the tools have been developed with focus on collaboration between architects. Software Architecture Warehouse (SAW) supports collaborative architectural decision making and enhances situational awareness by providing an environment for real-time synchronization of design spaces, voting, and discussion for software architects [30]. The purpose of the wiki-based tool $\mathrm{AD}_{\mathrm{kwik}}$ is also to support collaboration in decision making through sharing of knowledge about ADDs across project boundaries [31].

Although automated support is an important aspect in decision making, it is addressed very little by existing approaches. Ameller et al. propose Architech to support software architects during the architectural decision-making process by suggesting alternative decisions for satisfying non-functional requirements [32]. For this, optimization techniques are applied on an AK ontology. Also, ArchDesigner provides a quantitative quality-driven approach to find the best possible fit between various quality goals, competing architectural concerns and constraints [33]. It uses Multiple Attribute Decision Making (MADM) and optimization techniques to find a best-fitting architecture composed of interdependent architectural design decisions that satisfies prioritized quality goals. CoCoADvISE provides semi-automated support at the following steps: (1) decision making is guided through questionnaires and (2) semi-complete ADD documentations are generated based on suggested design solutions.

Similar to our tool, Zimmermann et al. provide an architectural design

method to combine pattern languages with reusable architectural decision models [6]. The goal of their approach is to provide domain-specific pattern selection advice and traceability links from platform-independent patterns to platform-specific decisions. However, the reusable ADD models have to be used manually by software architects while in CoCoADvISE automated decision guidance based on reusable decision models is provided.

The majority of the proposals have been evaluated either in case studies (e.g., [6, 26]), in industrial projects [25], or focus groups [30], and in a few cases no evaluation is reported. None of the tools has been empirically validated and only little feedback has been gathered by the users[2]. We conducted two controlled experiments with 171 software architecture students in total to test the supportive effect of the main concept of CoCoADvISE, namely the reusable ADD models, on capturing ADDs.

The reader can refer to Table 1 for an overview of the tools discussed in this subsection. In particular, we report on the type of the tool support (decision making or/and documentation), whether it provides automation support, and how it has been evaluated.

2.3. Other Empirical Studies Related to ADDs

There are various literature surveys and reviews on the approaches and tools for architectural decision making and documentation. For instance, Falessi et al. provide a comparison of various techniques for selecting architectural alternatives during decision making [7]. Shahin et al. provide a survey on existing ADD models and tools [18] while Bu et al. provide an analysis of decision-centric approaches for design support [36]. The mapping study by Tofan et al. provides an overview and taxonomy of the existing literature on ADDs [20]. Furthermore, Weinreich and Groher compare software architecture management approaches with focus on the main aims of documenting AK as well as the elements used to represent AK [37].

Except for these literature surveys and reviews, little empirical evidence (especially quantitative results) exists on the use and benefits of ADDs in the industry and by practitioners. Heesch et al. conducted a multiple-case study with four teams of software engineering students working in industrial projects in order to find out whether decision documentation supports novice architects in following a rational design process [38]. Shahin et al. test their hypothesis that the visualization of ADDs and their rationale improves the understanding of architecture designs in a controlled experiment with 10 participants [34]. Also, the supportive effect of pattern use in recovering of ADDs in terms of quality and quantity of documented ADDs has been validated in a controlled experiment with 33 software architecture experts and practitioners in a different study [12]. A few other qualitative studies such as [39, 40] focus on how software

[2]Except for two studies [30, 34] which contain only a very brief summary of the results and do not study their statistical significance.

Name	Decision Making	Documentation	Automation Support	Approach Evaluation
ArchPad [6]	+	−	−	Case study from the finance industry. SOA-related decisions (300) are documented.
ArchiTech [32]	+	−	Suggests alternative decisions based on NFRs	No evaluation reported
AD_{kwik} [31]	−	+	−	SOA-related decisions (300) are documented
Archium [28]	+	+	−	Motivating example
MAD 2.0 [26]	+	+	−	Case study (CRM system)
PAKME [24]	+	+	−	No evaluation reported
Compendium [27]	+	+	−	SOA-related decisions are documented
ADDSS [25]	+	+	−	Two industrial projects (multimedia system and virtual reality application)
ADVERT [29]	+	−	−	Evaluation using the Common Component Modeling Example (CoCoME)[1]
ArchDesigner [33]	+	−	Making trade-offs between stakeholders	Industrial project (Glass Box)
SAW [30]	+	−	−	In a focus group. 20 participants considered 100 issues with 5 alternatives each.
CoCoADvISE	+	+	Predefined questionnaires used during decision making	Two controlled experiments with 49 and 122 students respectively

[1] CoCoME is an example system which provides a benchmark for component-based modeling approaches [35].

Table 1: Comparison Overview of ADD Tools

architects make and document ADDs and which of them are being eventually documented. To the best of our knowledge, no empirical-grounded evidence exists yet on the impact of reusable AK and in particular reusable ADD models on the efficiency and effectiveness of software architects.

3. Architectural Decision Making Using CoCoADvISE

In this section, we introduce the CoCoADvISE prototype. In particular, we discuss its ADD meta-model (see Section 3.1) and core functionalities (see Section 3.2), and provide selected implementation details (see Section 3.3). CoCoADvISE (Constrainable Collaborative Architectural Design Decision Support Framework) provides semi-automated support for architectural decision making and documentation. In addition, it supports collaboration in decision making under various stakeholders' constraints, but this will not be discussed further in the current paper. The Web-based version of the tool was developed for the needs of the controlled experiments in such a way that our approach could be generalizable to a big extent. That is, the utilization of reusable ADD models, as well as the tasks – to be performed by the software architects – for making and documenting ADDs (see Section 4 for details) were designed in such a way that the obtained results can be generalizable and not bound to the use of the specific tool.

3.1. Reusable ADD Meta-model

CoCoADvISE provides tool support for modeling of reusable ADDs inspired by the Questions, Options, and Criteria (QOC) approach [23]. The CoCoADvISE ADD model extends the traditional QOC approach with – among others – additional dependencies between options, such as enforcement or inclusion, options and questions or decisions (i.e., an option triggers a next question or decision). In addition, it supports categorizing reusable solutions (i.e., options in QOC).

CoCoADvISE introduces a reusable ADD meta-model (see Figure 1) for the design space of certain application domains, consisting of *Decisions*, *Questions*, *Options*, and *Solutions*, as well as various relationships among them. For each design issue – that indicates a decision that has to be made given a specific design problem – a set of *Questions* (including one or more first *Questions*) providing multiple *Options* has to be modeled. Examples of relationships are that a selection of an *Option* triggers a follow-up *Decision* or an *Option* is incompatible with or enforces another *Option*. A selection of an *Option* may lead to a *Solution* to be suggested to the software architect. This *Solution* will be applied using an *Architecture Guidance*, that is a *Technology-related AK*, a *Design Pattern*, an *Architectural Pattern*, or a *Composite Reusable Solution* combining any of the aforementioned Architectural Guidance types.

The concepts that we introduce in CoCoADvISE are comparable to existing reusable ADD approaches such as the ones documented in [3, 9, 22]. Therefore, our hypotheses testing and evaluation results are considered to apply on similar approaches as well.

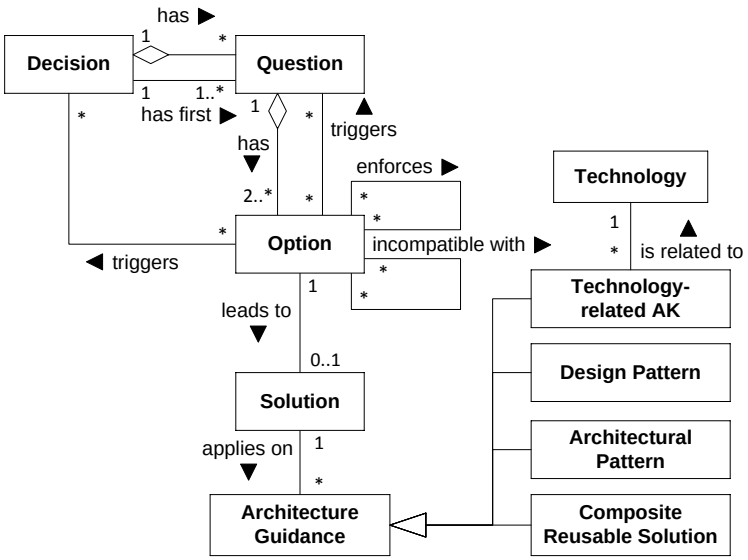

Figure 1: Reusable ADD Meta-model

3.2. Decision Support Based on Reusable ADD Models

The advantage of CoCoADvISE's reusable ADD models is that they are created only once for a recurring design situation and can be reused multiple times following the model-driven paradigm. In similar application contexts, corresponding questionnaires can be generated and used for making concrete decisions. Based on the outcomes of the questionnaires answered by software architects through the decision making process, CoCoADvISE can automatically resolve potential constraints and dependencies (e.g., reveal follow-on questions and decisions, deactivate incompatible options, etc.), recommend best-fitting design solutions, and eventually, generate half-filled ADD documentations. Using CoCoADvISE the software architect can choose from a list of reusable ADD models (see ① of Figure 2) and generate a questionnaire based on this model. The questionnaire provides architectural decision guidance through single-choice questions which lead to follow-on questions, decisions, and recommendations as indicated in ②. Finally, software architects may generate semi-complete documentations based on the CoCoADvISE recommendations such as the one shown in ③. CoCoADvISE fills in automatically some of the fields of the ADD documentations in template form [16]; architects need to fill in afterwards the rest of the required information.

3.3. Prototype Implementation Details

CoCoADvISE is the Web-based version of our previous Eclipse-based tool ADvISE [14] and was developed for the needs of the controlled experiments.

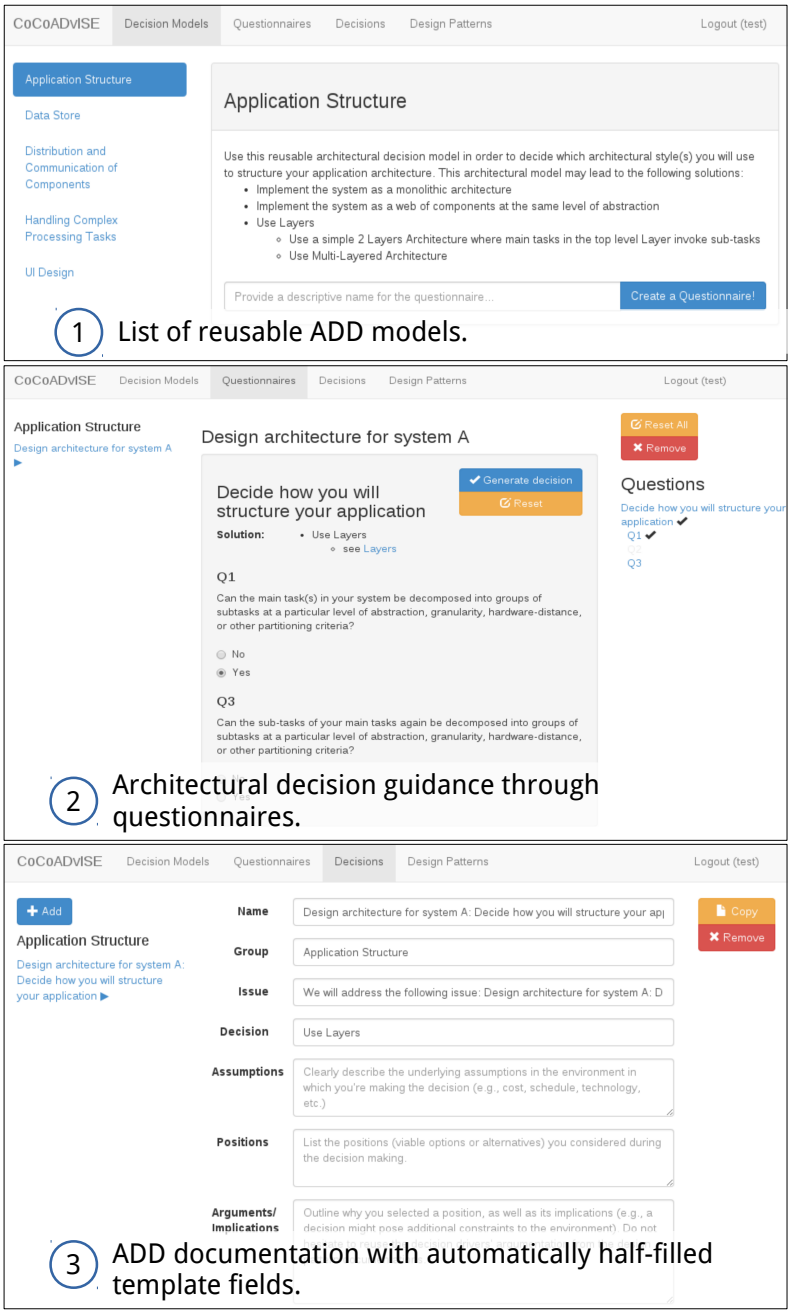

Figure 2: Screenshots of CoCoADvISE

It supports additionally collaborative architectural decision making. CoCoADvISE is a mostly client-side executed Single-page Web application that is implemented using Google's AngularJS[3] framework. The back-end of this Thin Server Architecture is founded on the Node.js[4] framework and runtime environment. In particular, it utilizes the real-time synchronization engine Racer[5] and persists the application state in a MongoDB[6] database.

4. Controlled Experiments

To measure the effect of using reusable architectural decision models on the efficiency and effectiveness of software architecture students we have conducted two separate controlled experiments with software engineering students. For the design and execution of the two experiments, we followed the guidelines of Kitchenham [41] while for the analysis, evaluation, and presentation of results we have consulted the advice of Wohlin et al. [42]. In the following subsections, we discuss the common goals and hypotheses of the controlled experiments and present their design and execution in detail.

4.1. Goals and Hypotheses

The goal of the experiments is to study and quantify the benefits of using CoCoADvISE and in consequence reusable ADD models for making and documenting ADDs in terms of quantity and quality related effectiveness, as well as time efficiency. For this, we compare two groups of students, one using reusable ADD models and one using an ad-hoc process based on pattern documentations to make and document ADDs in template form.

We postulate the following three null hypotheses and corresponding alternative hypotheses: Making and documenting ADDs using reusable ADD models...

H_{01} leads to lower or equal *quantity related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_1 leads to increased *quantity related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_{02} leads to lower or equal *quality related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_2 leads to increased *quality related effectiveness* of software architecture students compared to an ad-hoc process for architectural decision making.

H_{03} leads to lower or equal *time related efficiency* of software architecture students compared to an ad-hoc process for architectural decision making.

[3] `http://angularjs.org`
[4] `http://nodejs.org`
[5] `http://github.com/codeparty/racer`
[6] `http://mongodb.org`

H_3 leads to increased *time related efficiency* of software architecture students compared to an ad-hoc process for architectural decision making.

We perform statistical tests to find out whether the corresponding null hypotheses can be rejected. We expect that the users of CoCoADvISE will deliver ADDs of better quality and will manage to document more ADDs and in less time.

4.2. Parameters and Values

Several variables have been observed during the two experiments. In the following subsections, we discuss all dependent, independent, and derived variables we used for testing our hypotheses. A detailed list of variables (description, type, scale type, unit, and range) is provided in Table 2.

4.2.1. Dependent Variables

All dependent variables have been captured by and extracted automatically from CoCoADvISE's database. In particular, we instrumented its source code in such a way that we could precisely record all user activities within the system. The variable *time* indicates a single user's total time spent logged in in the application. The number of decisions that were documented by each student is indicated with the variable $numOfDecisions$. Finally, the variable *quality* refers to the quality of each ADD that was evaluated in a scale from 1 to 10 by two independent experts.

4.2.2. Derived Variables

To allow for a meaningful comparison of the *time* spent by the students to document the decisions we decided to introduce the variable $timeNorm$ which expresses the time needed per decision. In this way, we exclude the possibility that one treatment group needed less time to perform the tasks because they just delivered less work. In addition, we calculate the average points per student for the total of documented decisions ($qualityPerStudent$).

4.2.3. Independent Variables

The independent variables *group*, *exp* and $commExp$ can potentially influence the dependent variables. In particular, *group* indicates the participant's treatment group (i.e., control or experiment), and *exp* and $commExp$ refer to their programming experience in general and in the industry, respectively.

4.3. Experiment Design

The controlled experiments were conducted in the context of Information System Technologies and Software Architecture lectures respectively at the Faculty of Computer Science, University of Vienna, Austria. The first controlled experiment took place in Winter Semester 2013/2014 (January 2014) and the second in Summer Semester 2014 (June 2014). All participants were at the final semesters of their bachelor studies and had background in software architecture and design patterns. The readers can refer to Table 3 for a summary of

Type	Name	Description	Scale Type	Unit	Range
Dependent	*time*	Overall time needed to make and document decisions	Ratio	Minutes	Natural numbers including 0
	numOfDecisions	Number of documented decisions	Ratio	–	Natural numbers including 0
	quality	Quality of documented decision	Interval	–	1 (lowest) to 10 (highest)
Derived	*timeNorm* $(= \frac{time}{numOfDecisions})$	Time needed per decision	Ratio	Minutes	Natural numbers including 0
	qualityPerStudent $(= \frac{\sum_{i=1}^{numOfDecisions} quality}{numOfDecisions})$	Average quality of decisions	Interval	–	1 (lowest) to 10 (highest)
Independent	*group*	Treatment group	Nominal	–	Either "experiment" or "control"
	exp	Programming experience	Ordinal	Years	4 classes: 0-1, 1-3, 3-6, >6
	commExp	Commercial programming experience in industry	Ordinal	Years	4 classes: 0-1, 1-3, 3-6, >6

Table 2: Observed and Derived Variables

the experiment design in the two cases. We will refer to the first and second controlled experiment as *ORExp* (Online Retailer Experiment) and *UniISExp* (University Information System Experiment) respectively from the corresponding case studies that were used in each case.

	ORExp	**UniISExp**
Location	University of Vienna, Austria	University of Vienna, Austria
Time	January 2014	June 2014
Case Study	Online Retailer	University Information System
# Participants	49 (27 control / 22 exp.)	122 (56 control / 66 exp.)
# ADDs	319	762
# ADD Models	5 (16 patterns)	5 (40 patterns)

Table 3: Experiment Design Data

Participants. In the first controlled experiment in January 2014, from the 49 students of the lecture, 27 participated in the control group and 22 in the experiment group. In the second experiment in June 2014, a bigger sample of 122 students (56 in the control group and 66 in the experiment group) participated.

The experiments have been conducted in the course of mandatory practical exercises on architectural decisions for service-based software systems and distributed software systems respectively. The experiments took place in separate exercise groups (4 in the first and 6 in the second experiment) and in different computer labs at different times, which also explains the unequal number of participants in the corresponding control and experiment groups. The students in each group were assigned to the treatment groups randomly. In summary, 319 ADDs were documented in ORExp and 762 in UniISExp. All participants had experience with Java programming and design patterns and were familiar with the concepts of software architecture and architectural design decisions, as these topics had been already discussed in the corresponding lectures or were prerequisites to accomplish the previous exercises of the corresponding subjects.

Objects. A list of documented architectural design patterns and technology-related solutions was integrated in the CoCoADvISE tool for both groups in wiki format. The design patterns and architectural decision models were selected based on the lecture materials known to the students as well as the students' experiences from the previous programming exercises. The experiment group was provided additionally with a set of reusable architectural decision models and instructions how to use them for getting decision support with the aid of the CoCoADvISE tool.

Instrumentation. In the preparation phase, all students were asked to study the catalog of architectural design patterns and related technologies (afterwards integrated in the CoCoADvISE tool). Before starting with the experiment tasks,

all participants had to fill in a short questionnaire regarding their programming experience. Afterwards, the participants of both the control and experiment groups were provided with a description and requirements of the system to be designed. That was "An Online Retailer for Selling Books and Gadgets" and "A University Information System (IS)" for the OORExp and UniISExp experiments respectively. Both systems and their requirements were based on the experience of the authors with similar software systems from industrial projects. In Table 4 and Table 5 we give examples of the aforementioned software system's requirements. The complete descriptions of the design exercises can be found at `http://andromeda.swa.univie.ac.at/advise-experiment/exercises`.

Name	**Description**
Take Orders	Customers can place orders via two different channels: Web site and call center. Each of these systems is based on a different technology and stores incoming orders in a different data format. The call center system is a packaged application, while the Web site is a custom J2EE application. We want to treat all orders equally, regardless of their source. For example, a customer should be able to place an order via the call center and check the order status on the Web site. *Decide* how to deal with the different channels and feed the retailer with unified messages.

Table 4: Excerpt from the "Online Retailer for Selling Books and Gadgets" Requirements

Name	**Description**
Research Network	The University is collaborating with a central research system for establishing networks between researchers, dissemination of research results, access to publications, etc. The University IS should be able to send updates of new publications and research projects of its research staff. In this case, reliability and performance are not very important. The Research Network uses only the XML-RPC protocol for communication with external systems. *Decide* how you will design the communication between the University IS and the Research Network.

Table 5: Excerpt from the "University Information System (IS)" Requirements

The students had to consider six sets of requirements in OORExp (*Take Orders*, *Process Orders*, *Logging*, *Track Orders*, *Announcements*, and *Customer Login*) and also six sets of requirements in UniISExp (*General Requirements*, *Student Services*, *Services for Research and Administrative Staff*, *Library Service*, *Stream Service*, and *Research Network*) – concerning different parts of the software system – given in descriptive form. Additionally, some hints were provided with information about the concrete decisions that were expected (e.g.,

"Decide how to deal with the different channels and feed the retailer with unified messages" from Table 4). For each requirement, one or more ADDs were expected to be documented by the students.

Eventually, all participants were given access to the CoCoADvISE tool. The functionality of CoCoADvISE is described in Section 3 and the setting provided to the students can be viewed at `http://andromeda.swa.univie.ac.at/orexp` and `http://andromeda.swa.univie.ac.at/uniisexp` for ORExp and UniISExp respectively[7]. The participants of the experiment groups could reuse five ADD models which were different for the two controlled experiments. The names and descriptions of the provided ADD models are documented in Table 6 (for more details refer to the CoCoADvISE settings accessible at the aforementioned URLs.). In contrast, CoCoADvISE was modified for the participants of the control groups, so that the ADD model based support was hidden.

Blinding. In order to reduce the subjective bias of the participants and the reviewers we have applied double-blinding in both experiments. The participants were not aware of the two different treatment groups and different CoCoADvISE settings, therefore, they were not able to guess the goal of the experiments and whether they belong to an experiment or control group. In addition, the participants' documented ADDs were scrambled and given anonymized to the reviewers for evaluation. It was, thus, not possible to find out whether an ADD comes from the control or experiment group, which ADDs belong to which participants, and which ADDs belong together (i.e., were documented by the same person). Also, the reviewers did not get any other information about the goals and the different settings of the experiments.

4.4. Execution

As described in the previous section, the controlled experiments were executed in the context of the Information System Technologies and Software Architecture lectures (Faculty of Computer Science, University of Vienna) with students in the end of their bachelor studies, in Winter Semester 2013/2014 and Summer Semester 2014 respectively.

The practical course groups were randomly assigned to experiment and control groups. That is the reason why we have unequal groups of participants, namely 27 (control group) and 22 (experiment group) participants in ORExp and 56 (control group) and 66 (experiment group) participants in UniISExp. Nine students that participated in both experiments were excluded from the second experiment – although the topic and the tasks required in ORExp and UniISExp were different – in order to reduce the result bias. That led to a reduced sample of 50 and 63 participants for the control and experiment group of UniISExp respectively.

[7]To view and use the tools use the following user names (no password required): experiment or control for the experiment and control settings respectively.

ORExp	
External Interface Design	Use this reusable architectural decision model in order to decide how the service-based system will expose its interfaces to the external world.
Lifecycle Management	Use this reusable architectural decision model to decide how the lifecycle management of the remote objects (e.g., by creating per-request instances) and the resource consumption of the service-based system will be designed.
Message Routing	Use this reusable architectural decision model in order to make decisions on strategies for routing the messages in the service-based system.
Message Transformations	Use this reusable architectural decision model in order to make decisions on methods for transforming the messages inside the service-based system.
Service-based Invocations	Use this reusable architectural decision model to decide on the type of remote asynchronous invocations that will be used as well as the type of communication channels required for the exchange of messages.
UniISExp	
Application Structure	Use this reusable architectural decision model in order to decide which architectural style(s) you will use to structure your application architecture.
Data Store	Use this reusable architectural decision model in order to decide if and how you will need to store your data.
Distribute Components	Use this reusable architectural decision model in order to decide how you will distribute the different components of your system and how to design the communication between components.
Handling Processing Tasks	Use this reusable architectural decision model in order to decide how you will handle complex processing tasks required in your application.
UI Design	Use this reusable architectural decision model in order to decide how you will design the UI of your application.

Table 6: Reusable ADD Models Overview

In addition, before the execution of the controlled experiments we decided to exclude students that did not manage to achieve more than 50% of maximum points at the practical course. However, the final grade of the students at the practical course was not known at the time of execution, therefore, we excluded participants in the end of the semester and before we started with the analysis of the results. No students were excluded in ORExp while we had to exclude one student from the control group and two students from the experiment group in UniISExp. Thus, our sample for UniISExp was further reduced to 49 and 61 participants for the control and experiment group respectively.

Information about the programming experience of the students has been gathered with questionnaires that were given to the students in the beginning of the experiments. As we can see in Figure 3 and Figure 4 the programming experience, as well as the industry programming experience of the participants are quite similar for both groups. In ORExp, the control group has slightly more programming experience while in UniISExp we observe the opposite. The majority of the students in both experiments has 1 to 3 years of programming experience while some of the participants are quite experienced in programming (i.e., have more than 3 years of programming experience). However, only very few participants of both experiments have experience in industrial projects[8].

The exact same materials were handed out to all participants in the beginning of the exercise. As mentioned before, the experiment and control groups used different versions of CoCoADvISE, that is, the experiment group could use the reusable ADD models as shown in Figure 2 while this feature was deactivated for the control group. The participants had 90 minutes[9] time for reading, understanding, and performing the exercise. They were allowed to finish with the tasks earlier though. Access to the Web-based tool was given only during this session to avoid any offline work or discussion among the students. Apart from that, in order to avoid communication between the students or use of other materials the participants were prevented (i.e., using Web browser policies) from opening Web pages other than that of CoCoADvISE.

The collection of the participants' data was performed automatically during the experiment. In particular, all relevant information, such as created questionnaires, documented decisions, as well as all relevant events, such as deletions or modifications of architectural decisions, were saved in a database. In Appendix A we present six exemplary ADDs documented by the participants in the two controlled experiments along with their evaluations. The evaluation of the documented ADDs with the use of a Likert scale from 1 (lowest) to 10 (highest) was performed afterwards by two independent experts. The experts were instructed to consider the distances between the points of the 1–10 Likert scale to be constant. Hence, we argue that the corresponding variables *quality*

[8]Most of the students with industrial experience have been working from a couple of months to maximum one year at a company.

[9]This is a typical duration for a lab exercise in the two computer science courses. The experiments' tasks were thus designed given the limited time that the students would spend in the controlled environment.

and *qualityPerStudent* qualify as interval scale type. In case of high distances between the evaluations (i.e., 4 points or more between the ratings) the two experts discussed their evaluations until they reached a consensus. In total, 47 and 122 ADDs in ORExp and UniISExp respectively had to be revised. For the evaluation of the ADDs the following criteria were applied by both experts:

- The decision is stated clearly.
- The rationale of the decision is stated clearly.
- The decision corresponds to the described issue.
- The documented decision is a viable solution with regard to the described issue.
- Potential alternatives are documented.

Regarding the execution of the two controlled experiments no deviations from the initial study design occurred and no situations in which participants behaved unexpectedly.

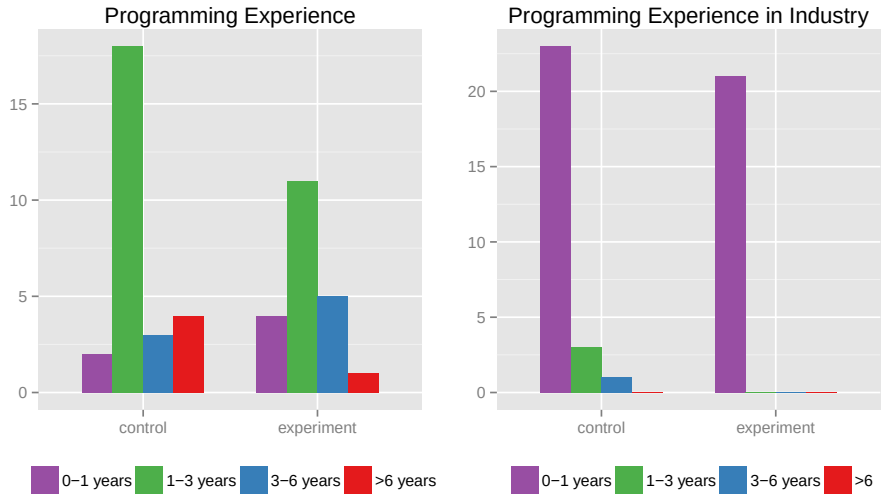

Figure 3: ORExp: Participants' Programming Experience

5. Analysis of Results

In order to analyze the data collected during the two controlled experiments and test our hypotheses (see 4.1) we used the R language and environment for statistical computing [43]. The raw data for these results are available at `http://andromeda.swa.univie.ac.at/advise-experiment/`.

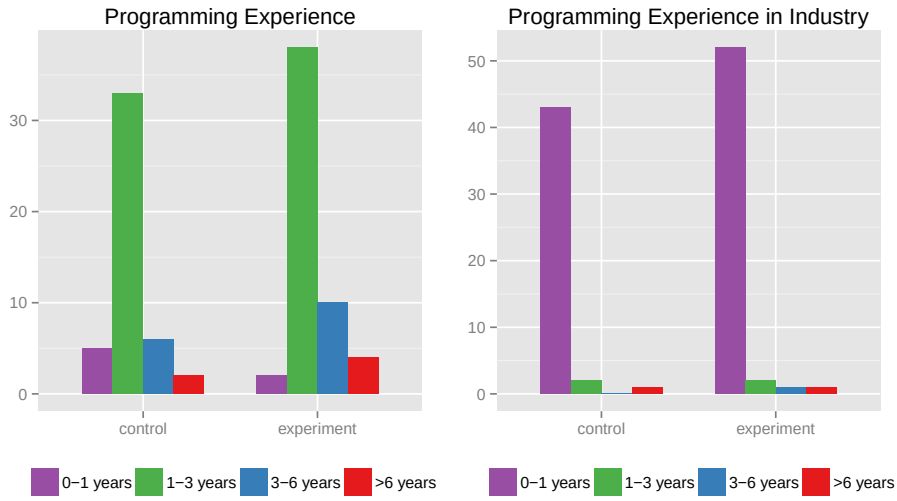

Figure 4: UniISExp: Participants' Programming Experience

5.1. Descriptive Statistics

As a first step in the analysis of the results we used descriptive statistics to compare the means and medians of the observed variables. In particular, Table 7 and Table 8 along with Figure 5 and Figure 6 display the mean and median values of the number of documented ADDs ($numOfDecisions$), the quality per ADD ($quality$), as well as the average quality of ADDs per student ($qualityPerStudent$), the time needed for completing the required tasks ($time$) and for documenting one decision ($timePerDecision$), for both control and experiment groups in the OORExp and UniISExp experiments respectively.

It is noticeable that the participants of the experiment group documented one ADD more on average than the control group: 7 against 6 and 6 against 5 for OORExp and UniISExp respectively. However, this is not necessarily an indicator of "higher efficiency" of the students as the experiment group may have needed more time for performing the tasks and thus has delivered more ADDs. We notice, though, that the control groups spent also more time on documenting a single decision, that is, 3.91 and 2.64 more minutes than the corresponding experiment groups on average. Therefore, the experiment groups needed *less time* to document *more decisions* although they have dedicated some of their time to understand and use the reusable ADD models. The reason is that CoCoADvISE with ADD model support generated semi-complete documentations which saved some time for the experiment group. We also calculated the average number of characters used for each ADD to find out whether the smaller number of ADDs and the increased time spent on their documentation was due to the smaller "size" of decisions. However, both treatment groups documented decisions of almost the same size in OORExp (489 characters for the control group

and 431 for the experiment group). We notice also a small difference in the length of the ADDs in UniISExp – 890 characters for the control group and 588 for the experiment group. This can be explained by the fact that often the participants of the control group reused parts of the pattern documentations to fill in some ADD template fields (e.g., Argumentation/Implications) which led to longer ADD texts – unlike for the experiment group these documentations provided the main source for decision support.

Variable	**Means**		**Medians**	
	control	**exp.**	**control**	**exp.**
$numOfDecisions$	5.59	7.57	6	7
$quality$	4.82	5.46	4.85	5.29
$qualityPerStudent$	4.9	5.35	4.5	5.5
$time$ (min)	44.71	35.75	48.11	36.69
$timePerDecision$ (min)	9.06	5.15	8.12	4.79

Table 7: Means and Medians of Observed Variables for ORExp

As mentioned in Section 4, each documented ADD was evaluated by two independent experts using a 10-point Likert scale ranging from very low quality (1) to very high quality (10). Likert scales are treated in the literature as both ordinal (e.g., [44]) and interval (e.g., [45]) scales. Ordinal scales show us the order, but not the distances between the ranking, that means, that in a 10-point Likert scale 2 means better quality than 1, 3 better quality than 2, and so on. Interval scales, on the other hand, show the order of things with equal intervals between the scale points. In order to be able to use descriptive statistics and statistical tests to analyze the quality of decisions we must treat the 10-point Likert scale as an interval scale. We assume, therefore, that this holds true in our case.

The average quality of the experiment group's ADDs is 5.46 and 6.96 compared to 4.82 and 6.34 in the control group for the ORExp and UniISExp respectively. In addition, the students in the experiment group delivered ADDs of better quality on average. To summarize, we would say that the treatment group that used the ADD models support *documented more ADDs* and achieved results of *better quality* and in *less time*.

On average, the participants of the experiment groups used 7 (ORExp) and 6 reusable ADD models. In addition, we calculated the time needed for filling in the questionnaires and the ADD templates separately. In ORExp 25% and in UniISExp 16% of the total time (1.07 and 0.81 minutes per questionnaire respectively) was spent on answering the questionnaires. The rest of the time was spent on filling in the semi-complete ADD documentations. Therefore, the effort for using the decision support capabilities of CoCoADvISE is very small in comparison with the effort needed to make and document decisions without automated or semi-automated guidance.

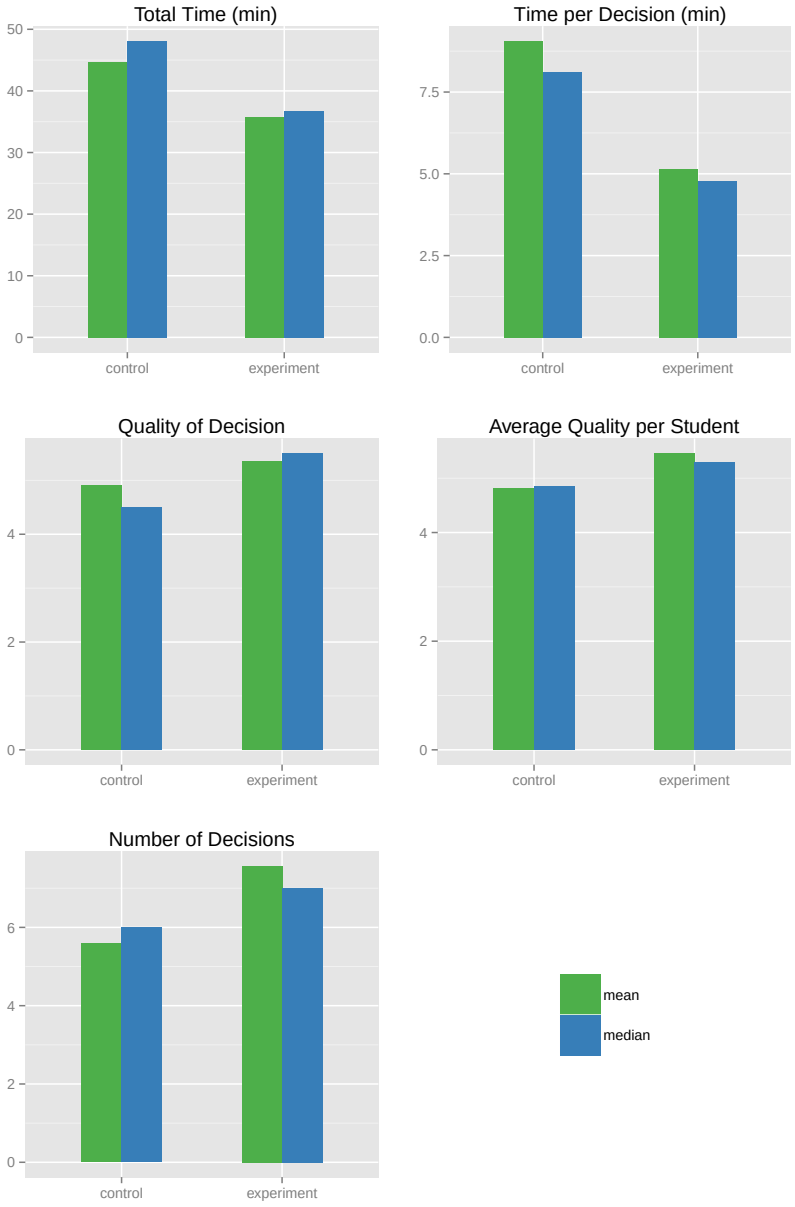

Figure 5: ORExp: Comparison of Means and Medians for the Observed Variables

5.2. Data Set Reduction

The few outliers that we noticed by studying the deviations from the means for each of the observed variables correspond to different participants, that is,

Variable	Means		Medians	
	control	**exp.**	**control**	**exp.**
numOfDecisions	5.70	6.72	5	6
quality	6.34	6.96	6.79	7.04
qualityPerStudent	6.37	6.92	7	7
time (min)	57.50	44.99	58.37	45.74
timePerDecision (min)	10.72	8.08	11.52	7.20

Table 8: Means and Medians of Observed Variables for UniISExp

these outliers correspond to ADDs documented by different students. Thus, the fact that these points have much higher or much lower values than the means (e.g., a student documented only a few decisions or needed much time for one decision) does not make necessarily the participant an outlier. Therefore, we excluded only students from the second controlled experiment (UniISExp) that had participated in the first as well and students who did not complete the practical course successfully (i.e., scoring less than 50% of the maximum points), and who therefore would make the study results vulnerable. This was done, however, before the data analysis (see explanation in Section 4.4); at this stage, we did not perform any further data set reduction.

5.3. Hypotheses Testing

5.3.1. Testing for Normal Distribution

Parametric tests like the *t*-test assume the normal distribution of the analyzed data. In order to decide whether to use parametric or non-parametric test for the analysis of the data, we applied the Shapiro-Wilk [46] normality test. The null hypothesis of the Shapiro-Wilk test states that the input data is normally distributed. We test the normality at a level of confidence of 95%. That means that if the calculated p-value is lower than 0.05 the null hypothesis is rejected and the input data is *not* normally distributed. Conversely, if the p-value is higher than 0.05, we can not reject the null hypothesis that the data is normally distributed.

Table 9 lists the p-values of the Shapiro-Wilk normality test for each observed variable and treatment group (all values are rounded to 4 decimal digits) for both controlled experiments. P-values that indicate normal distribution are emphasized (i.e., using **bold** font). We can see that only *numOfDecisions* and *qualityPerStudent* for OORExp and *time* for UniISExp controlled experiment exhibits a tendency of being normally distributed, while for the other variables we can not assume that they are normally distributed. As a result, we decided to pursue a non-parametric statistical test for analyzing the data.

5.3.2. Comparing the Means of Variables

To compare the means of the observed variables for the control and experiment groups, we applied the one-tailed Wilcoxon rank-sum test [47], a non-

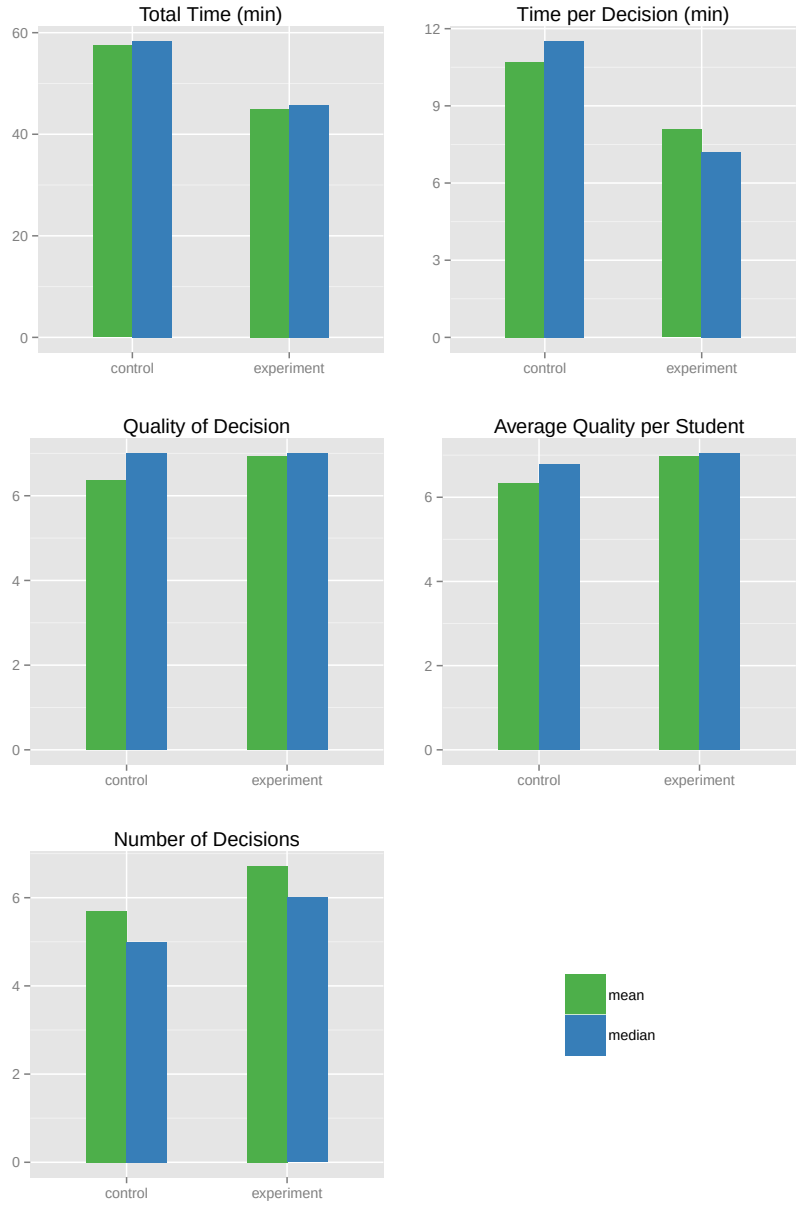

Figure 6: UniISExp: Comparison of Means and Medians for the Observed Variables

parametric test for assessing whether one of two data samples of independent observations is stochastically greater than the other. Its null hypothesis, which is appropriate for the hypotheses in our experiments, is that the means of the first

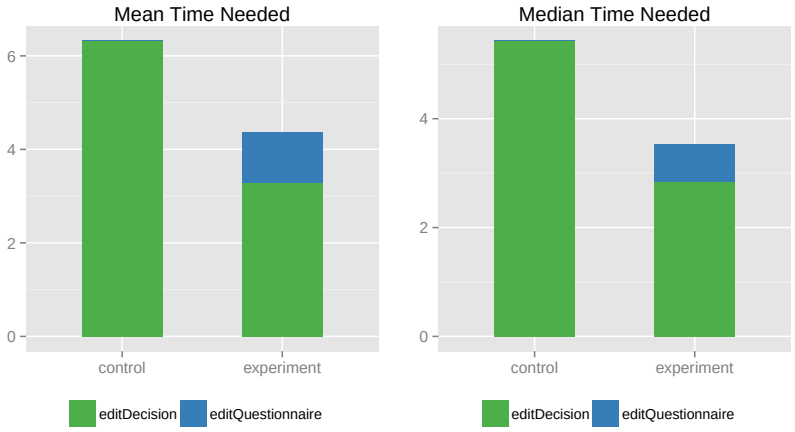

Figure 7: OORExp: Comparison of Time Spent by Participants for Making and Documenting ADDs

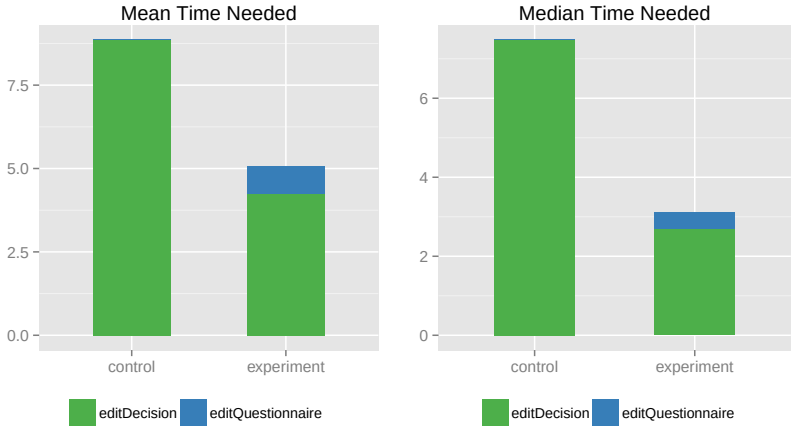

Figure 8: UniISExp: Comparison of Time Spent by Participants for Making and Documenting ADDs

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 (i.e., the level of confidence is 95%), 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.

Table 10 contains the p-values of five Wilcoxon rank-sum tests that were performed to find out whether we can reject the null hypotheses presented in

Variable	*OrExp* p-Value		*UniISExp* p-Value	
	control	**exp.**	**control**	**exp.**
$numOfDecisions$	**0.493**	**0.0941**	0.0009	0.0042
$quality$	$0(10^{-5})$	$0(10^{-5})$	$0(10^{-11})$	$0(10^{-9})$
$qualityPerStudent$	**0.7323**	**0.7145**	0.0046	**0.0643**
$time$ (min)	0.0107	**0.8832**	**0.3593**	**0.2717**
$timePerDecision$ (min)	0.002	**0.918**	**0.5181**	$0(10^{-10})$

Table 9: Shapiro-Wilk Normality Test

Section 4.1. Note that only the first four decimal places of the results are reported. Based on the obtained p-values, we can assess that almost all distributions show a statistically significant shift between each other and that most of the null hypotheses can be rejected. Analogously, Table 11 contains the p-values of t-Tests for those variables, where the assumption of normal distribution could not be rejected (see Table 9).

Testing Hypothesis $\mathbf{H_1}$. The experiment group documented significantly more ADDs than the control group. We reject the null hypothesis H_{01} (i.e., the use of reusable ADD models has no effect on the quantity related effectiveness of software architecture students) since the calculated p-value is 0.0027 (t-test: 0.0021) and 0.008 for the OrExp and UniISExp experiments respectively. Hence, there is evidence that the use of reusable ADD models for decision making and documentation increases the *quantity related effectiveness* of software architecture students.

Testing Hypothesis $\mathbf{H_2}$. In our experiments, we observed that the participants of the experiment groups delivered ADDs of better quality than the participants of the control groups. As this observation holds for both variables *quality* and *qualityPerStudent* we tested the null hypothesis H_{02} (i.e., the use of reusable ADD models has no effect on the quality related effectiveness of software architecture students) for these two variables. In the experiment OrExp, the p-values 0.0332 and 0.055 (> 0.05) do not allow us to reject H_{02} completely (however, t-test: 0.0330). We can reject this hypothesis for the quality of single ADDs but not for the average quality of documented ADDs per student. For UniISExp though, in which we tested a bigger sample of ADDs and participants, H_{02} could be rejected given the p-values 0.0459 and 0.0173 for the variables *quality* and *qualityPerStudent* respectively. Therefore, there is evidence for supporting H_2, that is, the ADDs of the experiment group were in total of better quality – according to the reviewers' evaluations – than those of the control group and the single participants' performance was also significantly "better". Hence, we can report evidence that using reusable ADD models for making and documenting decisions also increases the *quality related effectiveness* of its users.

Testing Hypothesis $\mathbf{H_3}$. Finally, we discovered that the experiment group needed significantly less time to document a decision. This holds also for the total time this group spent on the assigned tasks. With a p-value of 0.0045 and 0.0001 for *time* and *timePerDecision* in OrExp and corresponding values close to 0 (10^{-5}) in UniISExp we can reject the null hypothesis H_{03} for both experiments (the same holds for the corresponding t-test for the variable *time*), that is, the use of reusable ADD models has no effect on the time related efficiency of software architecture students. Thus, we can conclude that there is evidence that reusable ADD models lead to increased *time related efficiency* of software architecture students as well.

Hypothesis	**Variable** (μ)	**p-Value**	
(**Assumption**)		**OrExp**	**UniISExp**
$\mathbf{H_{01}}$ ($\mu_{exp} \geq \mu_{control}$)	$numOfDecisions$	0.0027	0.0080
$\mathbf{H_{02}}$ ($\mu_{exp} \geq \mu_{control}$)	$quality$	0.0332	0.0459
	$qualityPerDecision$	0.0550	0.0173
$\mathbf{H_{03}}$ ($\mu_{exp} \leq \mu_{control}$)	$time$	0.0045	$0(10^{-5})$
	$timePerDecision$	0.0001	$0(10^{-5})$

Table 10: Hypotheses Testing Results (Wilcoxon rank-sum Test)

Hypothesis	**Variable** (μ)	**p-Value**	
(**Assumption**)		**OrExp**	**UniISExp**
$\mathbf{H_{01}}$ ($\mu_{exp} \geq \mu_{control}$)	$numOfDecisions$	0.0021	
$\mathbf{H_{02}}$ ($\mu_{exp} \geq \mu_{control}$)	$qualityPerDecision$	0.0330	
$\mathbf{H_{03}}$ ($\mu_{exp} \leq \mu_{control}$)	$time$		$0(10^{-6})$

Table 11: Hypotheses Testing Results (*t*-Test)

6. Discussion

The following subsections discuss our main findings and their implications and inferences for software architects. We also report on the threats to validity.

6.1. Evaluation of Results and Implications

Increased Effectiveness. The first two hypotheses, i.e., H_1 and H_2, are related to the effectiveness of software architecture students which we study separately with regard to the quantity and the quality of the documented ADDs. As reported in Section 5, we have provided strong evidence for rejecting the corresponding null hypotheses H_{01} and H_{02}. Thus, reusable ADD models contribute both to the completeness and the quality of ADD documentations.

The semi-automated guidance using questionnaires provided by the CoCoADvISE tool allowed software architecture students (representative for novice software architects) to (1) identify the ADDs that had to be made, (2) understand the design space in the corresponding case study, (3) find out the alternatives for each design issue, and finally (4) document the appropriate design solution according to the requirements. Some of the participants of the experiment groups stated afterwards that CoCoADvISE's reusable models helped them find quickly the answer to their problem without needing to read all design pattern documentations. Especially, that turned out to be useful in cases where the design solutions were not so obvious from the requirements and required a better research and analysis of the design situation. For instance, for handling some complex processing tasks in the UniISExp case study, "single or multi threaded implementation of the tasks" would be viable solutions inferred by the reusable ADD models. Yet, most of the participants of the control group opted for a "pipes and filters" solution, which would clearly be "overkill" in this specific situation. Another phenomenon that we observed was that participants of the control groups were often not confident about what they should document in the ADD template fields. On the contrary, the reusable ADD models provided support (i.e., some fields are filled in automatically) and guidance (i.e., the reusable ADD models already contain design rationale) for filling in the decision related fields. We decided not to fill in further fields automatically (e.g., positions, arguments, etc.) as this would give a big advantage to the experiment groups and would make the comparison between the treatment groups difficult.

The observed phenomenon of experiment groups documenting more decisions can possibly be explained as follows. The reusable ADD models guided the students to follow-on decisions that were not considered by the participants of the control groups at all.

Systematically capturing and leveraging reusable AK in the form of reusable ADD models therefore may lead to increased effectiveness of architects. Our findings also validate Zimmermann et al.'s claim about some of the benefits of architecting with reusable ADD models: reusable ADD models (1) provide means for the semi-automatic decision identification and (2) improve the quality of decision making [8].

Increased Efficiency. We showed in the previous section that we accept our last alternative hypothesis H_3, that is, using CoCoADvISE's support on reusable ADD models reduces time and effort for software architecture students. This finding was highly expected though, as much "manual" work for making and documenting ADDs (e.g., reading documentations, filling in ADD templates repeatedly) is made redundant due to the semi-automated guidance by the questionnaires and the semi-complete documentations generated from decision recommendations. Thus, architects may need less time to document ADDs when guided by reusable ADD models.

6.2. Threats to Validity

To ensure the validity of our results, we consider the categorization of validity threats of Wohlin et al. [42] and discuss each of them separately in the context of our controlled experiments.

Conclusion Validity. The conclusion validity focuses on the relationship between the treatment we used in the experiment and the actual outcome, i.e., on the existence of a significant statistical relationship. The way we measured the working time of the students automatically from the database entries may pose a threat to conclusion validity, as the users might have spent some working time idle or with pen and paper. Apart from that, to measure the actual time spent on working with CoCoADvISE for performing the assigned tasks is very difficult, if not impossible, as the participants may have spent some time reading the tasks or familiarizing with the tool. However, we think that idle working times, times spent on other tasks, or offline work can largely be excluded due to the limited experiment time of 90 minutes in which the participants needed to work in a concentrated manner in order to get the work done.

In addition, the interpretation of the 10-point Likert scale that was used to rate the ADDs may pose a threat to the conclusion validity. In Section 5, we argued that we consider the Likert scale an interval scale, and thus the descriptive statistics and statistical tests that we applied are, making this assumption, valid. Another potential threat to validity is the subjectivity of the quality ratings of the reviewers. It could have happened as well that the one reviewer evaluated more strictly than the other. To reduce these risks, we asked two independent experts in the field of software architecture to evaluate all ADDs for both experiments and calculated afterwards the quality of each ADD on average. In case of disagreements in the evaluations the two reviewers needed to discuss their ratings until they reached a consensus. Nevertheless, this does not erase the subjectivity of the evaluations completely as some aspects related to the quality of ADDs like the evaluation of ADDs after the implementation of the software system are not taken into consideration in our case.

Internal Validity. The internal validity refers to the extent to which treatment or independent variables caused the effects seen on the dependent variables. In order to reduce this kind of validity threats, we made sure that the participants of both groups had at least medium experience in programming and design – with slight differences – and that they were aware of the architectural design patterns they had to use for making and documenting architectural decisions. For this, the participants were asked to study all related patterns in advance. Apart from that, the students had applied some of the architectural design patterns that appeared in the controlled experiments in previous practical assignments (i.e., in programming exercises concerning the implementation of various software systems).

The experiments were carried out in controlled environments and the treatment group members were in different rooms. Thus, they could not know the goals of the experiment, as they were not aware of the existence of different

groups or/and the different CoCoADvISE settings for the two treatment groups. During the experiment, the students were allowed to use only the CoCoADvISE tool and all other applications and Web pages were blocked. This way, we prevented access to other Internet materials as well as the participants' communication through chat applications. Also, an observer in the room ensured that no interactions between the participants of the same room occurred to ensure that the students worked individually.

We also prevented any access to CoCoADvISE and the accompanying materials outside the dedicated session or outside the labs where the controlled experiments were carried out.

Construct Validity. The construct validity focuses on the suitability of the experiment design for the theory behind the experiment and the observations.

The variables that have been observed in the experiment are regarded accurate and objective as they are related to the actual use of tools and were automatically extracted from the database. The students did not have any training or experience with the tools. A potential threat to validity may be posed by improper use of CoCoADvISE by participants who did either not understand the purpose of the reusable ADD models and the questionnaires or did not comprehend the tasks. We also need to take into account language difficulties in some cases (CoCoADvISE texts were in English, all other materials were both in English and German). We tried to overcome such problems by encouraging the students to ask the instructors for further explanations. Another potential threat to construct validity is the fact that only one measure was used to evaluate the quality of the documented ADDs, which does not allow us to cross-check the experiments' results.

Finally, there is a potential threat to validity imposed by the participating subjects, knowing the research topics of our research group and therefore being able to "guess" results that we hoped to obtain. We tried to minimize the impact of this threat by choosing independent and external experts that were not aware of our research agenda.

External Validity. The external validity is concerned with whether the results are generalizable outside the scope of our study. The subjects of the experiments had medium programming experience and were familiar with the ADDs they were asked to make. However, only few students had experience in the industry. We hence consider the participants of both controlled experiments to be representative for novice software developers or architects. A threat to validity of the relevance of our study's findings in practice is the use of students instead of practitioners in both experiments. We tried to mitigate the threat by educating the students on the patterns and architectural decisions used in the experiment. At the time of the experiments, all participants had theoretical knowledge of, as well as programming experience with the majority of the design solutions that had to be considered for the required tasks. Therefore, to a certain extent, our results can be transferred to novice software architects. However, we will need to conduct similar experiments with practitioners in order to be able to

consider our experiment results generalizable – applicable to professional novice (or more experienced) software architects.

Also, the fact that the treatment groups used the CoCoADvISE tool only, poses the threat that the results are specific only for this tool. However, we tried to implement in CoCoADvISE general concepts and practices from reusable ADD models [3] and ADD documentations [16]. Hence, we expect the same results if the participants worked with tools based on similar concepts as well.

As mentioned before, the measurements were extracted from the tool database, avoiding any bias by the experimenters.

6.3. Inferences

van Heesch et al. found out that the quality of ADDs increases when focus is set on the use of design patterns for documentation [12]. Our findings confirm and supplement these previous results for students of software architecture. That means that the use of design patterns in the form of reusable ADD models further increases the effectiveness of students leading to less effort and better quality for ADD documentations. We claim that our results can be transferable to novice/trainee software architects, that needs to be proven, however, in an industrial context with practitioners.

Although we have conducted our experiments with students with software architecture background and with little commercial experience, we believe that our results can apply similarly to novice or more experienced architects in the industry. To validate these claims we will need more empirical evidence and feedback from the use of reusable architectural decision models in practice and in the context of industrial projects.

In practice, ADDs remain either undocumented or partially documented and the documentations are often dispersed in different places like wikis, meeting minutes, and issue tracking systems [40]. In addition, documenting ADDs that correspond to recurring design issues is tedious and time-consuming. Providing the means for semi-automated decision making and documentation based on reusable ADD models may encourage software architects to capture AK regularly and systematically. That will contribute to reducing AK vaporization [2].

It is of course important that reusable ADD models that have been documented in the literature (e.g., [8, 9, 21, 22]) and have been tested in a very limited scale (i.e., by practitioners of the same company, etc.) get validated and enriched by practitioners from different domains. That will increase the acceptance of such reusable AK and confirm Nowak et al.'s vision of collaboration and knowledge exchange between different AK repositories (i.e., repositories containing reusable architectural decision models) [48]. Therefore, the usefulness of tools like CoCoADvISE which provide semi-automated decision support based on such ADD models will get more significant.

7. Conclusions and Future Work

In this paper, we reported the results of two separate controlled experiments on architectural decision making and documentation based on reusable ADD

models. Due to the current lack of empirical evidence, the experiments were designed to determine the supportive effect of reusable ADD models for software architects. We were particularly interested in quantifying the impact of reusable ADD models on a software architecture student's efficiency and effectiveness while making and documenting decisions. Hence, we implemented and instrumented a custom-made tool (CoCoADvISE) for making and documenting ADDs, that tries to incorporate the best practices of current (i.e., state of the art) ADD tools, in such a way that each action in the tool was timestamped and stored in a database. This valuable information allowed us to extract and deduce efficiency and effectiveness related metrics.

Our experiments provided strong evidence that reusable ADD models can potentially increase both the effectiveness and the efficiency of software architecture students while making and documenting ADDs. We can further report, that our findings are in line with similar studies (see, e.g., [12]) and support the claims regarding reusable ADDs (see, e.g., [8, 9, 21, 22]) in principle. We consider our results to be applicable for novice software architects as well. Thus, as part of our ongoing future work, we will repeat our experiments with different reusable ADD models and different types of participants. In particular, we strive for experimenting with practitioners to see if our assumptions and results are still valid in industrial contexts. At the same time, we plan to further investigate the educational aspect of reusable ADD models. To that end we want to find out if and to what extent reusable ADD models are conducive for novice software architects to learning how to systematically use patterns.

Acknowledgments

We would like to thank the reviewers who evaluated the architectural decisions in both controlled experiments. We also thank all students for participating in the experiments.

Appendix A. Examples of Documented ADDs

We present in this appendix exemplary ADDs documented by the participants of the two controlled experiments along with the experts' ratings for these decisions. In particular, we include three decisions from ORExp and three decisions from UniISExp.

Name	D01: Use Fire and Forget for Logging
Group	Communication/Logging
Issue	All messages that the online retailer exchanges are sent to a logging system that stores them in a database for 1 year.
Decision	Fire and Forget Pattern
Assumptions	Loss of logging messages is acceptable.
Positions	Our logging service is implemented as a remote object. Recording of log messages must not influence the execution of the client and loss of individual log messages is acceptable. In this case Fire and Forget can be used.
Arguments/ Implications	A client application wants to notify a remote object for an event. Neither a result is expected, nor does the delivery have to be guaranteed. A one-way exchange of a single message is sufficient. -> simple, non-blocking communication
Related Decisions	-

Table A.1: Example 1 from OORExp / Reviewer 1: 7, Reviewer 2: 8

Name	Process Orders - Multiple single orders
Group	Messaging
Issue	Inventory systems can only process single orders.
Decision	Splitter
Assumptions	The Splitter splits the order into individual order items. Each message refers to the original message (e.g., the order ID) so that the processing results can be correlated back to the original message.
Positions	-
Arguments/ Implications	A Splitter can break large messages into individual messages to simplify further processing. In our Systems this needs to be done because the inventory systems can only process single orders. The only downside is that the overhead gets bigger because of the correlation identifiers.
Related Decisions	Process Orders - Routing and Take Orders.

Table A.2: Example 2 from OORExp / Reviewer 1: 7, Reviewer 2: 5

Name	OrderMessageLocationProcessor-ADD1: Message processing
Group	Message Routing and Processing
Issue	OrderMessageLocationProcessor ADD1: Message processing
Decision	Use a message splitter
Assumptions	The order processing sever need to receive orders from different locations, the call center and the website. The order processing server needs to know where is one order coming from and treat all orders equally.
Positions	Message splitter with Message aggregator
Arguments/ Implications	Each order message contents the location information and the order information, before processing the order, a splitter is needed to split the order message. As the retailer wants to treat each order equally, an aggregator is not needed here.
Related Decisions	OrderConfirmMessageRouter-ADD2: Message routing/filtering

Table A.3: Example 3 from OORExp / Reviewer 1: 10, Reviewer 2: 9

Name	Decision 02 S-T
Group	Streaming
Issue	Streams should be kept on a central server and be downloaded on demand, bandwidth and transformation in different audio/video formats is a problem.
Decision	I have chosen asynchronous streaming where the sequence of the packets is important, but not the time.
Assumptions	Because of the high amount of students it is important to keep the bandwidth low when streaming. Because of that, the streaming should be asynchronous, because it is faster and needs less bandwidth as synchronous or isochronous streaming.
Positions	Alternatives would be synchronous or isochronous streaming, which are "better" but also need a lot more resources, especially when streaming to multiple clients like it is done at the university.
Arguments/ Implications	Streaming offers high scalability as it requires less memory usage at the server side. Also, it is simple to implement and maintain. However, it cannot support transactional processing or interaction between different clients. And as I chose the asynchronous streaming I can decrease the memory usage even more.
Related Decisions	none

Table A.4: Example 4 from UniISExp / Reviewer 1: 9, Reviewer 2: 9

Name	Communication between the University-IS and e-learning
Group	Communication between the uni-IS and the e-learning
Issue	Current system: bad communication between the uni-is and the e-learning and the information not in time.
Decision	Shared repository
Assumptions	Invocation parameters are inefficient for large data sets also bad if the information varies from invocation to invocation.
Positions	Very often data needs to be shared between components. In sequential architectures the only way to share data between the components is to use invocation parameters but this is inefficient for large data sets. Also it might be inefficient, if the shared information varies from invocation to invocation. Finally the long-term persistence of the data requires a centralized data management. Thus, Shared Repository is used as a central data store, accessed by all other independent components. It offers suitable means for accessing the data, for instance, a query API or language and is scalable to meet the clients' requirements. It must ensure data consistency and must handle problems of resource contention, for example by locking accessed data. It might also introduce transaction mechanisms. A Shared Repository maintains the common data on which the application's components operate, the components themselves access and modify the data in the shared repository, and the state of the data in the Shared Repository instigates the control flow of specific components.
Arguments/ Implications	The Shared Repository pattern increases the level of abstraction in the code. This may decrease understandability for developers who are unfamiliar with the pattern. Although implementing the pattern reduces the amount of redundant code, it generally increases the number of classes that must be maintained. The Shared Repository pattern helps to isolate both the service and the list access code. Isolation makes it easier to treat them as independent services and to replace them with mock objects in unit tests. Typically, it is difficult to unit test the repositories themselves, so it is often better to write integration tests for them. If the data is being cached in heavily loaded systems, performance can be an issue.
Related Decisions	Faster, better for large data and long term persistence of the data requires, suitable for accessing the data

Table A.5: Example 5 from UniISExp / Reviewer 1: 6, Reviewer 2: 5

Name	D01: Architectural style for distribution
Group	System Design
Issue	System needs to be deployed on different machines
Decision	Design of a service-oriented architecture
Assumptions	(1) System needs to be scalable and deployed onto several different machines. (2) Heterogenous components like e-learning platforms (that change very fast), research networks and library components need to be integrated in such a way that changing or replacing one of the subsystems does not lead to a huge effort in reprogramming the core application.
Positions	A poor alternative to this approach would be to implement each functionality on its own and just access a shared repository where input/output data is stored.
Arguments/ Implications	Service-based architecture is a very well established pattern and meets all the requirements for this project. As there is a very loose coupling between the communicating entities, it is no problem to replace or add subsystems which is very crucial for such a dynamic system.
Related Decisions	-

Table A.6: Example 6 from UniISExp / Reviewer 1: 8, Reviewer 2: 9

References

[1] A. Jansen, J. Bosch, Software Architecture as a Set of Architectural Design Decisions, in: 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, PA, USA, IEEE Computer Society, 2005, pp. 109–120.

[2] N. B. Harrison, P. Avgeriou, U. Zdun, Using Patterns to Capture Architectural Decisions, IEEE Software 24 (4) (2007) 38–45.

[3] O. Zimmermann, T. Gschwind, J. Küster, F. Leymann, N. Schuster, Reusable Architectural Decision Models for Enterprise Application Development, in: 3rd International Conference on Quality of Software Architectures (QoSA), Medford, MA, USA, Springer, 2007, pp. 15–32.

[4] R. Farenhorst, R. Izaks, P. Lago, H. Van Vliet, A Just-In-Time Architectural Knowledge Sharing Portal, in: Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA), 2008, pp. 125–134.

[5] M. Galster, M. A. Babar, Empirical Study of Architectural Knowledge Management Practices, in: IEEE/IFIP Conference on Software Architecture (WICSA), 2014, pp. 239–242.

[6] O. Zimmermann, U. Zdun, T. Gschwind, F. Leymann, Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method, in: 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), Vancouver, BC, Canada, IEEE Computer Society, 2008, pp. 157–166.

[7] D. Falessi, G. Cantone, R. Kazman, P. Kruchten, Decision-making Techniques for Software Architecture Design: A Comparative Survey, ACM Computing Survey 43 (4) (2011) 33:1–33:28.

[8] O. Zimmermann, J. Koehler, L. Frank, Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design, in: D. Lübke (Ed.), Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture (SEMSOA 2007), Hannover, Germany, 2007, pp. 46–60.

[9] I. Lytra, S. Sobernig, U. Zdun, Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study, in: Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA-ECSA'12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 111–120.

[10] S. Abrams, B. Bloom, P. Keyser, D. Kimelman, E. Nelson, W. Neuberger, T. Roth, I. Simmonds, S. Tang, J. Vlissides, Architectural Thinking and Modeling with the Architects' Workbench, IBM Systems Journal 45 (3) (2006) 481–500.

[11] J. F. Hoorn, R. Farenhorst, P. Lago, H. van Vliet, The Lonesome Architect, Journal of Systems and Software 84 (9) (2011) 1424–1435.

[12] U. van Heesch, P. Avgeriou, U. Zdun, N. Harrison, The supportive effect of patterns in architecture decision recovery – A controlled experiment, Science of Computer Programming 77 (5) (2012) 551–576.

[13] O. Zimmermann, Architectural Decisions as Reusable Design Assets, IEEE Software 28 (1) (2011) 64–69.

[14] I. Lytra, H. Tran, U. Zdun, Supporting Consistency Between Architectural Design Decisions and Component Models Through Reusable Architectural Knowledge Transformations, in: Proceedings of the 7th European Conference on Software Architecture (ECSA), ECSA'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 224–239.

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

[16] J. Tyree, A. Akerman, Architecture Decisions: Demystifying Architecture, IEEE Software 22 (2) (2005) 19–27.

[17] L. Lee, P. Kruchten, Capturing Software Architectural Design Decisions, in: 2007 Canadian Conference on Electrical and Computer Engineering, IEEE Computer Society, 2007, pp. 686–689.

[18] M. Shahin, P. Liang, M. R. Khayyambashi, Architectural design decision: Existing models and tools, in: IEEE/IFIP Conference on Software Architecture/European Conference on Software Architecture, IEEE, 2009, pp. 293–296.

[19] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, M. A. Babar, A comparative study of architecture knowledge management tools, Journal of Systems and Software 83 (3) (2010) 352–370.

[20] D. Tofan, M. Galster, P. Avgeriou, W. Schuitema, Past and future of software architectural decisions – A systematic mapping study, Information and Software Technology 56 (8) (2014) 850–872.

[21] U. Zdun, M. Strembeck, Reusable Architectural Decisions for DSL Design: Foundational Decisions in DSL Development, in: Proceedings of 14th European Conference on Pattern Languages of Programs (EuroPLoP 2009), Irsee, Germany, 2009, pp. 1–37.

[22] C. Mayr, U. Zdun, S. Dustdar, Reusable Architectural Decision Model for Model and Metadata Repositories, in: F. de Boer, M. Bonsangue, E. Madelaine (Eds.), Formal Methods for Components and Objects, Vol. 5751 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 1–20.

[23] A. MacLean, R. Young, V. Bellotti, T. Moran, Questions, Options, and Criteria: Elements of Design Space Analysis, Human-Computer Interaction 6 (1991) 201–250.

[24] M. A. Babar, I. Gorton, A Tool for Managing Software Architecture Knowledge, in: Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent, SHARK-ADI'07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 11–.

[25] R. Capilla, F. Nava, J. C. Duenas, Modeling and Documenting the Evolution of Architectural Design Decisions, in: Proceedings of the Second Workshop on SHAring and Reusing Architectural Knowledge Architecture, Rationale, and Design Intent, SHARK-ADI'07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 9–.

[26] A. Zalewski, S. Kijas, D. Sokołowska, Capturing Architecture Evolution with Maps of Architectural Decisions 2.0, in: Proceedings of the 5th European Conference on Software Architecture, ECSA'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 83–96.

[27] M. Shahin, P. Liang, M. R. Khayyambashi, Improving Understandability of Architecture Design Through Visualization of Architectural Design Decision, in: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK'10, ACM, New York, NY, USA, 2010, pp. 88–95.

[28] A. Jansen, J. V. D. Ven, P. Avgeriou, D. K. Hammer, Tool Support for Architectural Decisions, in: Proceedings of the 6th working IEEE/IFIP Conference on Software Architecture, IEEE Comp. Soc., 2007, pp. 4–4.

[29] M. Konersmann, Z. Durdik, M. Goedicke, R. H. Reussner, Towards Architecture-centric Evolution of Long-living Systems (the ADVERT Approach), in: Proceedings of the 9th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA'13, ACM, New York, NY, USA, 2013, pp. 163–168.

[30] M. Nowak, C. Pautasso, Team Situational Awareness and Architectural Decision Making with the Software Architecture Warehouse, in: Proceedings of the 7th European Conference on Software Architecture, ECSA'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 146–161.

[31] N. Schuster, O. Zimmermann, C. Pautasso, ADkwik: Web 2.0 Collaboration System for Architectural Decision Engineering, in: Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE), Knowledge Systems Institute Graduate School, 2007, pp. 255–260.

[32] D. Ameller, O. Collell, X. Franch, ArchiTech: Tool Support for NFR-guided Architectural Decision-Making, in: Requirements Engineering Conference (RE), 2012 20th IEEE International, 2012, pp. 315–316.

[33] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, B. Benatallah, A Quality-driven Systematic Approach for Architecting Distributed Software Applications, in: 27th International Conference on Software Engineering, ICSE'05, ACM, New York, NY, USA, 2005, pp. 244–253.

[34] M. Shahin, P. Liang, Z. Li, Architectural Design Decision Visualization for Architecture Design: Preliminary Results of A Controlled Experiment, in: Proceedings of the 1st Workshop on Traceability, Dependencies and Software Architecture (TDSA), ACM, 2011, pp. 5–12.

[35] S. Herold, H. Klus, Y. Welsch, C. Deiters, A. Rausch, R. Reussner, K. Krogmann, H. Koziolek, R. Mirandola, B. Hummel, M. Meisinger, C. Pfaller, CoCoME - The Common Component Modeling Example, in: The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007], 2007, pp. 16–53.

[36] W. Bu, A. Tang, J. Han, An Analysis of Decision-centric Architectural Design Approaches, in: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK'09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 33–40.

[37] R. Weinreich, I. Groher, A Fresh Look at Codification Approaches for SAKM: A Systematic Literature Review, in: Proceedings of the 8th European Conference on Software Architecture (ECSA), ECSA'14, Springer-Verlag, Berlin, Heidelberg, 2014, pp. 1–16.

[38] U. van Heesch, P. Avgeriou, A. Tang, Does decision documentation help junior designers rationalize their decisions? A comparative multiple-case study, Journal of Systems and Software 86 (6) (2013) 1545–1565.

[39] C. Zannier, F. Maurer, A qualitative empirical evaluation of design decisions, ACM SIGSOFT Software Engineering Notes 30 (4) (2005) 1–7.

[40] C. Miesbauer, R. Weinreich, Classification of Design Decisions: An Expert Survey in Practice, in: Proceedings of the 7th European Conference on Software Architecture, ECSA'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 130–145.

[41] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, J. Rosenberg, Preliminary Guidelines for Empirical Research in Software Engineering, IEEE Trans. Softw. Eng. 28 (8) (2002) 721–734.

[42] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[43] R. C. Team, et al., R: A language and environment for statistical computing, R foundation for Statistical Computing.

[44] G. Vigderhous, The level of measurement and 'permissible' statistical analysis in social research, Pacific Sociological Review 20 (2) (1977) 61–72.

[45] H. R. Maurer, Todd J.; Pierce, A Comparison of Likert Scale and Traditional Measures of Self-Efficacy, Journal of Applied Psychology 83 (2) (1998) 324–329.

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

[47] H. B. Mann, W. D. R., On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other, Annals of Mathematical Statistics 18 (1) (1947) 50–60.

[48] M. Nowak, C. Pautasso, O. Zimmermann, Architectural Decision Modeling with Reuse: Challenges and Opportunities, in: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK'10, ACM, New York, NY, USA, 2010, pp. 13–20.