Science of Computer Programming 77 (2012) 551–576

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

The supportive effect of patterns in architecture decision recovery—A controlled experiment

Uwe van Heesch [a,b,*], Paris Avgeriou [a], Uwe Zdun [c], Neil Harrison [a,d]

[a] University of Groningen, Groningen, The Netherlands
[b] Fontys University of Applied Sciences, Venlo, The Netherlands
[c] Faculty of Computer Science, University of Vienna, Austria
[d] Utah Valley University, Orem, UT, USA

ARTICLE INFO

Article history:
Received 29 June 2011
Received in revised form 22 November 2011
Accepted 24 November 2011
Available online 7 December 2011

Keywords:
Software architecture
Architecture decisions
Recovery
Controlled experiment

ABSTRACT

The documentation of software architectural design decisions is important to help people understand the system and the rationale behind architectural solutions. In practice, the documentation of such decisions is regularly done after the fact, or skipped completely. To support software maintenance and evolution, the decisions have to be recovered and described. This is often hindered by the fact that the original architects are not available any more, or they do not completely remember the reasons for making the decisions. Additionally, the whole process is very expensive. In this paper, we hypothesize that architecture decision recovery can be more efficient by focusing on recovering decisions related to applying architecture patterns. To test this hypothesis, we designed a controlled experiment that was conducted to analyze the impact of architecture patterns on the quality and quantity of architecture decisions recovered after the fact. We are able to provide statistical evidence that a focus on patterns significantly increases the quality of decisions, while no conclusive evidence concerning the quantity of decisions was found.

1. Motivation

During the architecting phase of a software project, many decisions are made that influence the fundamental structure and behavior of the software system to develop. The architects responsible for making these decisions have to take into consideration the concerns of the most important stakeholders, quality attribute requirements, architecturally-significant functional requirements and constraints that limit the potential outcome of the decisions. Architecture decisions satisfy some of the concerns while they may potentially violate others. As a consequence, architecting involves negotiations between stakeholders and making trade-offs between different requirements and concerns that have to be satisfied during the software design. The perfect solution often does not exist; the rationale of architecture decisions explains the related trade-offs and optimizations.

While architects consciously and subconsciously make these decisions, they regularly neglect to document them appropriately [16]. In some cases, the outcome of decisions is represented in architecture documentation, various UML design diagrams, or at the very least in the source code; however, the exact problem that is solved, the concerns that were balanced, and the rationale behind the decision are usually omitted [37].

* Corresponding author at: University of Groningen, Groningen, The Netherlands.
E-mail addresses: uwe@vanheesch.net (U. van Heesch), paris@cs.rug.nl (P. Avgeriou), uwe.zdun@univie.ac.at (U. Zdun), Neil.Harrison@uvu.edu (N. Harrison).

0167-6423/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.11.008

After some time, when the project advances, even the architect who originally made the decisions will have difficulties remembering all the details and eventually the knowledge gets lost to a great extent. In the literature, this problem is called *architectural knowledge vaporization* [14,16,20,37].

This phenomenon becomes especially problematic when software systems are maintained or extended. During software evolution, developers must understand the existing system well in order to make informed decisions on changes and extensions. New requirements and changing system behavior make it necessary to carefully review the original architecture decisions before making additional ones. In absence of decent project documentation, the architectural knowledge and especially the past decisions have to be recovered. Otherwise, new architecture decisions may conflict or override existing ones, or may repeat past mistakes. Therefore, recovering architecture decisions is of paramount importance for a successful system evolution.

Unfortunately, recovering architecture decisions presents several challenges. If the project documentation is poor, the recovery of architecture decisions is a resource-intensive task that requires a lot of experience and has a high risk for ambiguity and misunderstandings. If architecture decisions were not explicitly documented by the original architects, the new software development team responsible for making changes to the system typically has to rely on the running application, the source code, incomplete textual documentations, end-user manuals, and fragmentary or out-of-date design diagrams (e.g., in UML).

It is often challenging enough to identify architectural solutions and the corresponding decisions on the basis of these artifacts. Finding out why they were chosen, which requirements and concerns they satisfy, and which consequences the implementation of the approaches has, requires vast knowledge and experience from the analysts. To make matters worse, most approaches cannot be seen as isolated solutions; they are pieces of a larger puzzle that only make sense if they are examined in the architecture as a whole.

One way to efficiently recover architecture decisions is to look for patterns applied in the architecture and reuse their extensive documentation (that can be found in the pattern literature) in the context of the system under study. Patterns typically describe the problem space, in which they are applicable and give advice for applying the solution they propose. In that respect, architectural patterns capture many important aspects of architecture decisions [14]; the same aspects that are subject to architectural knowledge vaporization. Once the pattern is identified, the pattern description can be used to explore the pattern's problem space, the consequences of applying it, related decisions, and possible trade-offs the original architects made. Of course, not all architecture decisions are related to applying patterns; but some of the most important ones are. We assume that architecture decision recovery based on patterns is more efficient than ad-hoc, intuitive decision recovery.

The goal of this paper is to empirically validate whether architecture decision recovery is more efficient regarding the quality and quantity of architecture decisions, if the recovery focuses on identifying applied patterns. Specifically, we intend to answer the following research question: Are the quality and quantity of recovered architecture decisions higher if the recovery focuses on identifying applied architectural patterns than in the general case?

To answer the research question, we conducted a controlled experiment during the European Conference on Patterns Languages of Programs (EuroPLoP) [15] in July 2009 and during a software architecture workshop for industrial practitioners in Venlo, the Netherlands in April 2011. In total, 33 software engineering experts from academia and from the industry took part. They were asked to recover architecture decisions on the basis of an architectural documentation of the JBoss J2EE application server [21]. Half of the participants were explicitly asked to focus on identifying patterns in the architecture, while the other half was told to rely on their experience and intuition when performing the recovery. The data from the experiment was analyzed, and the quality and quantity of the recovered architecture decisions were compared.

The results of the experiment provide strong evidence for the benefits of using patters concerning the quality of recovered decisions. The study did not provide conclusive evidence concerning the quantity of decisions.

The rest of this paper is organized as follows: Section 2 presents related work. Section 3 explains the design of the controlled experiment including the introduction of variables and hypotheses, while the next section presents details about the execution of the experiment. We analyze the results of the study and present the hypotheses testing in Section 5. Section 6 contains an interpretation of our findings, a discussion of threats to validity, and finally observations and lessons learned. Section 7 concludes and presents future work.

2. Related work

The design of this experiment and the theoretical background of the hypotheses presented in Section 4 are related to multiple research areas: software architecture, architecture recovery and software patterns. Within architecture recovery, we distinguish between architectural reconstruction and architecture decision recovery.

The former concerns the reconstruction of an architecture, which was never documented by the architects, or whose documentation is no longer synchronized with the system "as-is" [4]. The latter mostly focuses on recovering the decisions regarding architectural solutions and the reasoning behind the latter; especially concerning the satisfaction of requirements. It does not only answer the question *what* the architecture is like, but also *why* it is like that. Thus, architecture decision recovery is complementary to architectural reconstruction. The following paragraphs discuss the related work in each of the aforementioned areas.

Many definitions exist for software architecture. In ISO/IEC/IEEE 42010 [18], the international revision of IEEE Std 1471-2000 [17], the architecture of a system is defined as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". In the software architecture literature, it is also described as the result of making a set of design decisions that impact the overall structure and behavior of a software system [6,19,37]. These decisions are usually called architecture decisions. While some approaches mainly document the outcome of these architecture decisions in different architectural views (e.g. [8,18,27], others focus on documenting the decisions themselves [34,37].

However, these approaches focus on the documentation of software architecture during the architecting process, or at least by the architect himself. They primarily concern forward engineering scenarios and try to conserve knowledge that is or might become useful in the future. In contrast to this, during architecture recovery, this knowledge, and often also the people possessing it, are partially or totally unavailable.

Several approaches exist that mainly use source code as a basis for architecture reconstruction [23,26]. Krikhaar et al. propose an approach that uses the source code and naming and coding conventions to extract the architecture of a system ex post [26]. Although the original architects, if available, can make a contribution to this process, the goal is not to recover rationale, but to expose the architecture of a system in suitable representations to allow impact analyses on quality attributes and to incrementally improve the architecture. Kazman and Carrière present an approach to reconstruct architecture that centers around "Dali", a suite that integrates multiple tools to extract and analyze software architecture [23]. Again, the approach involves the extraction of possibly multiple models from the source code and other programming artifacts that describe the architecture including elements, relationships, and attributes of relevant entities. These models can be seen as different views on the architecture.

In contrast to the static source code analyzes approaches presented above, dynamic system analysis focuses on the runtime behavior of systems. Yan et al. describe an approach called "DiscoTect" [39]. The architecture of a running system is analyzed using state-machines to identify common patterns of runtime behavior in monitored system events. As a result, DiscoTect identifies applied architectural styles, whose runtime patterns have been defined in state machines. Other outcomes of the method are different views representing the architecture of the system at runtime.

In all aforementioned approaches, and mainly in all other architecture reconstruction approaches [25], the source code, system events and other artifacts used to configure and run the system are automatically processed using tool support to build different types of models that represent the architecture of the analyzed system as-is. They do not aim at recovering the problem space of the architectural constructs or even the rationale behind them. Architecture decisions are not made explicit.

Jansen et al. present an approach to recover architectural design decisions [20]. The described method involves reconstructing detailed designs and several architectural views on a level of abstraction that is suitable to recover architecture decisions. Source code and information from the original architects form the basis for the architecture reconstruction. This approach has a different focus than the previously mentioned ones, as representations of the architecture are solely created for the purpose of recovering architecture decisions. Depending on the purpose of the models, they can remain on a high level of abstraction. In contrast, this paper focuses on the recovery of architecture decisions based on existing architecture documentation that was created manually and is not necessarily a result of source code analysis. Additionally, the recovery of decisions is done manually and profits from interpretation skills that humans have in contrast to machines. This makes the approach applicable for situations, in which the documentation neither involves multiple architectural views, nor a detailed design of the whole system, but fragmentary textual descriptions that include box-and-line diagrams and other sketches of aspects of the system. Moreover, it explicitly supports the recovery of the problem space and the rationale behind decisions, not just the specifically applied solutions.

Software architectural patterns,[1] like all patterns, capture generic solutions to recurring problems in specific contexts [1,7,10]. They provide reusable architectural knowledge for a particular application domain [30]. Architectural patterns reason about design alternatives, consequences, and trade-offs concerning software qualities, which are performed when applying them [7].

Especially architectural patterns explicitly discuss the consequences of their usage concerning the quality attributes of the target architecture and mention related patterns [7].

A comparison of patterns and architecture decisions is presented in [14]. In this paper, the advantages of documenting patterns applied in a software architecture are explicated. Other approaches exist that also make use of patterns as source of architectural knowledge [3,36,40,41]. However, all presented approaches propose to document the usage of patterns during the architecting process, while this paper focuses on using patterns in architecture decision recovery, where large parts of the original reasoning is not explicitly available any more.

3. Design of the experiment

For the design of the experiment, the guidelines by Kitchenham et al. [24] and Wohlin et al. [38] were used. The former present general guidelines for software engineering experimentation and give some advice concerning the design, execution,

[1] In the remainder of this article, for simplicity, we will use the word pattern meaning software architectural pattern.

Table 1
Dependent variables.

Description	Scale type	Unit	Range
Quantity of recovered decisions	Ratio	Decisions	Positive natural numbers including zero.
Quality of recovered decisions	Interval	N.A.	Five point Likert-scale. One for *very poor*, Five for *very high*.

analysis, and presentation of empirical studies without going into detail. The latter present the phases in more detail, discuss statistical tests and their suitability for different types of studies. In this experiment, Kitchenham et al.'s guidelines were primarily used in the planning phase of the experiment, while Wohlin et al.'s advice was used as a reference for the analysis and interpretation of the results. Jedlitschka's and Pfahl's reporting guidelines [22] are used to describe the experiment in this paper. The following subsections of the proposed template were left out, because they were not applicable, or the content was already presented in other sections: Inferences are discussed Section 6; relation to existing evidence is presented in Section 2; impacts of the approach on time and quality are discussed in Section 5; interpretation and general limitations of the study are discussed in Section 6.2. Note that the usage of this template introduces a certain level of redundancy, because a distinction between the design and the actual execution of the experiment is made. Some subsections of the execution phase are similar to corresponding subsections of the design phase.

3.1. Goal, hypotheses, parameters, and variables

The goal of the experiment is to find out, if architecture decision recovery that is based on systematic identification of patterns in the architecture leads to higher quality or quantity of recovered decisions compared to architecture decision recovery that is performed ad hoc and intuitively. Although systematic approaches for architecture decision recovery exist (e.g. see [20]), practitioners in the industry still perform recovery in an ad-hoc, intuitive way.

The study goal led to the following null hypotheses and corresponding alternative hypotheses:

H_{01}: Focusing on identifying patterns in architecture decision recovery leads to lower or equal *quality* of recovered decisions compared to ad-hoc, intuitive recovery.

H_1: The *quality* of recovered decisions is higher when the recovery focuses on identifying patterns in the architecture, compared to ad-hoc, intuitive recovery.

H_{02}: Focusing on identifying patterns in architecture decision recovery leads to lower or equal *quantity* of recovered decisions compared to ad-hoc, intuitive recovery.

H_2: The *quantity* of recovered decisions is higher when the recovery focuses on identifying patterns in the architecture, compared to ad-hoc, intuitive recovery.

3.1.1. Dependent variables

Two dependent variables were observed during the experiment, as shown in Table 1: the quality and the quantity of recovered architecture decisions.

The following aspects are taken into consideration to measure the quality of the recovered decisions:

- Problem/Issue: The architectural design issue that is addressed by the decision.
- Decision: The outcome or solution imposed by the decision.
- Alternatives: Possible alternative solutions addressing the design issue.
- Arguments: A justification for the chosen decision instead of the alternatives.
- Requirements: Functional and non-functional requirements that are satisfied or affected by the decision.
- Related Decisions: Decisions that are related to or imposed by the current decision.

The quality of the recovered decisions was assessed by two independent experts in the field of software architecture (later also referred to as *analysts*) using a five point Likert-scale [33] that ranges from one for *very poor* to five for *very high* quality. The ratings were left to their own experience and interpretation, but they were asked to take the aforementioned aspects of decision quality into consideration.

Quantity of architecture decisions is defined as the number of recovered architecture decisions. Decisions that both analysts concordantly rated as non-architectural would be excluded from the analysis. As it is hard to clearly estimate in how far a design decision is critical to a system design and hence concerns the architecture of the system, we provided some examples of architecture decisions and left further evaluation to the expertise of the analysts.

3.1.2. Independent variables

The goal of the experiment was to discover the influence of patterns on the quality and quantity of decisions obtained from architectural recovery. Therefore, two different treatments were defined for the participants. One group of participants was explicitly told to focus on identifying patterns in the architecture documentation; the participants in the other group did not get any specific advice, but they were allowed to perform this task as they would normally do it. The first group is referred to as *pattern group*, the latter as *control group*.

Table 2
Independent variables.

Description	Scale type	Unit	Range
Group	Nominal	N.A.	Possible values: *Pattern group*, *Control group*.
Affiliation	Nominal	N.A.	Possible values: *university/academia*, *industry*, *other*
Programming experience	Ordinal	Years	4 classes: 0, 1–3, 3–7, >8
Architecture experience	Ordinal	Years	4 classes: 0, 1–3, 3–7, >8
Middleware experience	Ordinal	Years	4 classes: 0, 1–3, 3–7, >8
Frequency of pattern usage in projects	Ordinal	Percent	4 classes: 0%, <25%, >25%, 100%
Number of well known patterns	Ordinal	Patterns	4 classes: <5, 5–10, 11–20, >20

Table 2 shows other variables that could have an influence on the dependent variables. They relate to characteristics of the participants and mainly concern previous experience. In the design of the study, these variables were eliminated by defining blocking rules to balance the characteristics among the pattern group and the control group.

3.2. Experimental design

To test the hypotheses, we conducted two executions of a controlled experiment [5] using exactly the same study design. The first execution took place at EuroPLoP 2009 [15]; the second execution took place during a software architecture workshop at the Fontys University of Applied Science in Venlo, the Netherlands, in April 2011.

3.2.1. Participants

The schedule of the EuroPLoP conference, where the first execution of the study took place, had reserved time slots for so called *focus groups* (FGs). Attendees were able to propose topics in advance and publish them on the conference website. Multiple FGs were scheduled concurrently, so participants had to make a selection. We announced a focus group in advance and stated explicitly that we were planning to do an experiment on architecture recovery based on an existing architecture documentation. The participation in the focus group was voluntary, but all participants had to take part in the experiment. It was assumed that the primary motivation for taking part in the experiment was personal interest in architecture recovery. We expected to have 10 to 15 participants, based on experience from former focus groups. A background in at least one software-engineering discipline was presumed.

The second execution during the software architecture workshop was announced as a practical session on architecture decision recovery. The participation in the workshop was free. Invitations were sent to alumni students from the hosting university of applied science, and colleagues from their companies. We assumed that the greatest part of the participants would have a significant industrial background and expected between 15 and 30 attendees.

The experimental design described in the following subsections was followed in the same way in both executions of the experiment.

3.2.2. Object

The basis for the architectural recovery was a five page document about the JBoss J2EE application server version 2.2.4, an excerpt of a research article on the JBoss architecture written by Jenny Liu at the University of Sydney in April 2002 [28].

JBoss, in the described version, is a free open source application server implementing the J2EE specification. The documentation does not explicitly mention the usage of any pattern, but hints exist in form of component names. The name *RequestBroker* for example hints at the usage of the Broker pattern [7].

The document describes the conceptual architecture and lists technologies and frameworks used in the implementation. Besides text, some box-and-line diagrams are used to illustrate components, and control- and data-flow in parts of the architecture. The participants received a print-out of the document.

The architecture of the used JBoss server is dominated by a microkernel, which was implemented using the Java Management Extension (JMX). The major JBoss services are encapsulated in so called MBeans, which are managed by an MBean server that is part of JMX. JMX itself has a layered architecture; the agent layer contains the MBean server. The bottom layer communicates directly with the Java virtual machine. Please refer to [28] for the detailed description of the architecture.

3.2.3. Blocking

To be able to explicitly analyze the influence of patterns in architecture recovery, we split the participants in two groups. One group was asked to identify and document architecture decisions related to patterns, whereas the other group did not get corresponding advice. The goal was to reduce the effect of independent variables that might influence the results of the analysis.

Because of the rather small sample size, we decided not to assign the participants to the groups randomly, but to balance the groups explicitly based on affiliation (university, industry, other), programming experience (0 years, 1–3 years, 3–7 years, 8 or more years), architecture experience (0 years, 1–3 years, 3–7 years, 8 or more years) and experience with object-oriented middleware (0 years, 1–3 years, 3–7 years, 8 or more years).

Table 3
Instrumentation overview.

Phase	Instrument	Purpose
Introduction	First questionnaire	Gather information needed for blocking
	Example decision	To explain the concept of architecture decisions
Experiment	Blank decision templates	Used by the participants to document the recovered decisions
	Pattern catalog	Provided to the pattern group as pattern reference
Wrap-up	Second questionnaire	Gather information needed for interpretation and validation of the results

3.2.4. Instrumentation

Table 3 shows an overview over the instruments used in the three phases of the experiment.

In the introduction phase, we asked all participants to fill in a questionnaire prior to the recovery exercise to gather information needed to perform the blocking (affiliation, programming experience, architecture experience and middleware experience). Unique random numbers were attached to the questionnaires to identify the participants throughout the experiment. They were also mapped to every recovered architecture decision.

In the same phase, we introduced the concept of architecture decisions to all attendees and presented one elaborate example on how to recover and document a decision based on a small part of an architecture documentation. The example decision was handed out to all participants, so they could use it as a guideline during the experiment. The template used to document the decision was taken from [34].

In the next phase, the participants were asked to document recovered decisions based on the same template. Therefore, we handed out as many blank templates on paper as needed by the participants. Some fields in the template were optional, whereas *Problem/Issue*, *Decision*, *Arguments*, and *Related Requirements* were marked as mandatory fields. We encouraged the participants to provide as much information as possible regarding at least the mandatory fields.

Every member of the pattern group additionally received a printed copy of the most well-known architectural patterns [2,7].

Using the catalog to identify patterns was optional. It was assumed that many of the participants had knowledge about architecture patterns anyway. However, the catalog was provided to serve as a reference and as a reminder for the participants to focus on patterns. Any patterns or architectural styles were allowed that were used to solve an architectural problem. As described in Section 3.1.1, we left it up to the analysts to judge, whether a recovered decision was architectural, or not.

An additional questionnaire (later also referred to as second questionnaire) was designed to gather further information from the participants after the experiment in the wrap-up phase. It contained questions concerning previous experience with software patterns and the usefulness of patterns during the recovery. Although this data is not needed for testing the hypotheses, it is useful for the interpretation and validation of the results. We asked these questions *after* the experiment for two reasons. First, because the questions regarding patterns could have influenced the participants of the control group (the non-pattern group) prior to the experiment; they could have guessed that patterns play an important role in the other group, and consciously or unconsciously also focus on patterns. Second, because we are interested in the way the participants actually performed the recovery.

3.2.5. Blinding

To eliminate subjective bias on the part of the participants and the experimenters, double-blinding was applied in the experiment. Although the participants had to realize that there are two different groups, they were not able to understand the purpose of the group division, the difference in treatments and if they belong to the experimental group or the control group. To prevent the experimenters from being biased, the participants handed in the results using a participant number that does not allow to draw conclusions on their real identity. The participant numbers were assigned to them on the first questionnaire.

Because the experimenters necessarily know which participant number belongs to which group, the quality ratings of the results were done by two independent experts. We asked two people from our own professional network to do the analysis.

Table 4 summarizes the relevant experience of the analysts. The data was gathered using a web questionnaire.

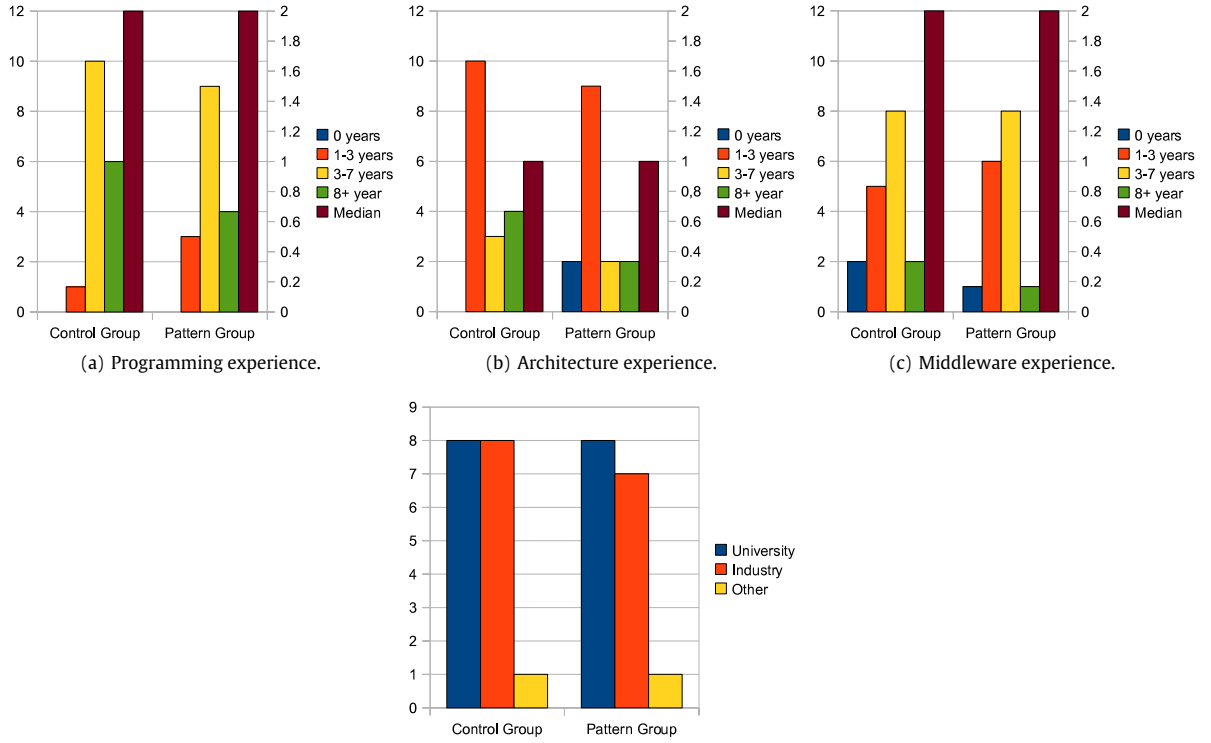
These analysts did not get any specific information about the experiment. They were just asked to rate the quality of some documented decisions on a scale from one to five as described in the variables section. Before handing the decisions out to the analysts, we pseudonymized them a second time by attaching a unique random number to every decision and internally mapping it to the participant number. That way it was impossible for the analysts to find out which decision belongs to which participant, which decisions belong together, and which decisions belong to the pattern group. As mentioned earlier, the fact that there were two groups was not communicated to the analysts either.

3.2.6. Data collection procedure

After 30 min. of introduction and grouping, the participants started with the recovery. The provided templates had to be used to document the recovered architecture decisions on paper. The participants of the groups were distributed over two separate rooms according to the group membership. Two experimenters were present in each room to answer questions

Table 4
Characteristics of the analysts.

Characteristic	Analyst 1	Analyst 2
Working experience in the industry	9 years	33 years
Experience in the field of software architecture	9 years	12 years
Experience with object-oriented middleware like J2EE	5 years	3 years
Involved in making architecture decisions (5-point Likert-scale from *very frequently* to *very rarely*)	very frequently	frequently
Involved in documenting architecture decisions (5-point Likert-scale from *very frequently* to *very rarely*)	very frequently	frequently
Involved in the analysis of architecture decisions (5-point Likert-scale from *very frequently* to *very rarely*)	very frequently	frequently



(a) Programming experience. (b) Architecture experience. (c) Middleware experience.

(d) Affiliation.

Fig. 1. Distribution of participants.

related to the instructions and to take care that participants did not communicate. Once the session was completed, the filled-in decision templates were collected by the experimenters and finally a wrap up session was planned to collect comments on the experiment and to fill in the questionnaires about pattern experience and pattern usage mentioned in Section 3.2.4. Including a 30 min. break, the experiment lasted three hours.

4. Execution

4.1. Sample and Preparation

As described in the design section, the experiment was announced as a focus group during EuroPLoP 2009 and as a practical session on architecture decision recovery at the workshop in Venlo.

At EuroPLoP, twelve people were willing to take part in the experiment, from which one had to be rejected because of a lack of software engineering experience. Twenty-two people took part in the practical session in Venlo, from which none had to be rejected.

All participants filled in the first questionnaire and were afterward assigned to either the pattern group or the control group. The blocking procedure went as expected according to the experimental design.

Fig. 1 shows previous experience and affiliation of the participants, as assigned to the pattern group and the control group. The figures accumulate the data from all participants from the two executions of the experiment.

Subfigures (a) to (c) show the previous experience of the participants concerning programming, architecture and middleware. Additional to the total numbers of participants in each class, median values are shown for each group. The medians are aligned to the right vertical axis, whereas all other values are aligned to the left vertical axis. Medians were calculated as follows. First, each of the year-intervals was assigned to a single value in an ordinal scale ranging from zero to three. '0 years' was assigned to the value 0, '1–3 years' was assigned to 1 and so on. Then medians were calculated based on the ordinal scale. In total, the median programming-, architecture- and middleware experience for both groups is two, which means that the participants in the groups were balanced concerning their previous knowledge.

At the same time, the participants from both groups were introduced to the concept of architecture decisions using the prepared recovered decision. The introduction took approximately 15 min.

4.2. Data collection performed

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

In the workshop execution in Venlo, one of the participants from the control group did not hand in the recovered decisions after the experiment. Consequently, his data could not be taken into consideration. Other than that, everything went as planned.

4.3. Validity procedure

The experiment took place in a controlled environment. The participants were assigned to two different rooms according to their group (pattern, or control group). At least one experimenter was present in each room during the whole experiment time to assure that participants did not use forbidden material and did not talk to each other. After the experiment, all filled-in decision templates were collected by the experimenters before any of the participants left the room. There were no situations in which participants behaved unexpectedly.

5. Analysis

5.1. Descriptive statistics

We use descriptive statistics to visualize the collected data as a first step in the analysis. The first two subsections are related to the hypotheses tests: Section 5.1.1 presents an analysis of the quality of documented decisions. Section 5.1.2 concerns the quantity of decisions. The last section presents an analysis of the data gathered in the second questionnaire, in which the participants were asked about their previous experience and the usefulness of patterns during architecture recovery. The results are compared to those of the analysis of the quality and quantity of the recovered decisions.

5.1.1. Quality of recovered decisions

As explained in the design section, the quality of knowledge in every recovered decision was rated by two independent experts using a five point Likert-scale ranging from one for *very poor quality* to five for *very high quality*.

The level of scaling (e.g. nominal, ordinal, interval, ratio) for Likert-scales is hard to determine. It is common sense that Likert-scales are at least ordinal in nature [12]. For the quality ratings in this experiment this is given. A decision that was rated with five has a higher quality than a decision rated with four. However, to be able to use parametric statistical tests like the t-test, at least an interval scale [31] character of the scale must be assumed. This is the case if equal distances between the points on the scale can be assumed, e.g. the difference between the ratings five and four would be the same as the distance between ratings two and three. In our specific case we assume that this holds true.

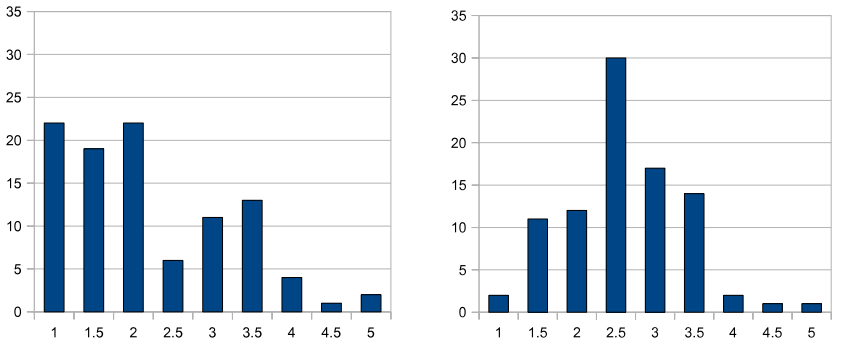
In the analysis of the experiment, a t-test for independent variables [29] is used to calculate the significance of the found results. Levene's significance test is used to find out whether equal variances of the quality ratings can be assumed.

Based on the data in Table 9 in the appendix, the following descriptive statistics apply for the quality of knowledge. Fig. 2 shows the frequency of rated quality for the decisions recovered by the pattern group and the control group. From the figure, we see that the quality of decisions in the pattern group seems to be higher than the quality of decisions in the control group. Moreover, it is noticeable to see that the most frequent quality ratings in the pattern group are 2.5 (33.3%) and 3 (18.9%) compared to 1 (22.0%) and 2 (22.0%) in the control group.

As argued before, we interpret the Likert-scale as an interval scale; so the mean, standard deviation, variance, and range apply as measures. Additionally, the median value is calculated, which would also be applicable for ordinal scales. Table 5 shows a comparison between the statistics for the control group and the pattern group. Besides the fact that the average quality of decisions in the pattern group is higher than in the control group, it is noticeable to see that the variance in the control group is much higher than the variance in the pattern group. This means that the dispersion of quality ratings is higher in the control group.

5.1.2. Quantity of recovered decisions

The quantity of recovered decisions is measured counting all architecture decisions that were not excluded as being non-architectural by both analysts. One of the analysts excluded some decisions as being non-architectural, whereas the other



(a) Absolute distribution Control Group. (b) Absolute distribution Pattern Group.

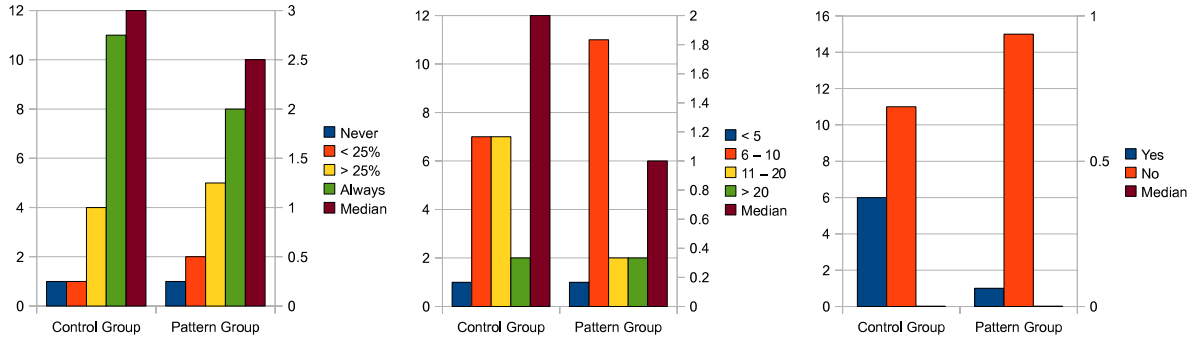
Fig. 2. Frequency: quality of decisions.

Table 5
Additional descriptive statistics.

	Control group	Pattern group
N	100	90
Mean	2.185	2.611
Std Dev	1.032	0.752
Variance	1.064	0.566
Median	2.000	2.500

Table 6
Descriptive analysis of the quantity of decisions.

	Control group	Pattern group
N	16	15
Mean	6.25	6.0
Std Dev	4.386	3.359
Variance	19.267	11.286



(a) How often do you apply patterns in projects? (b) How many patterns do you know well? (c) Have you done architecture recovery before?

Fig. 3. Pattern and architecture recovery experience.

analysts did not exclude decisions at all. Because there was no mutual agreement on any of the cases to be excluded, we included all decisions in the quantitative analysis.

The number of participants in the two groups is shown in Table 6 together with the mean values, the standard deviations, and the variances for the respective group. The mean quantity of decisions, which is measured in terms of the number of recovered architectural decisions, is slightly higher in the control group than in the pattern group. The standard deviation in the control group is considerably higher than in the pattern group.

5.1.3. Analysis of second questionnaire

In this section, the results from the second questionnaire, which was filled in after the experiment took place, will be presented.

Fig. 3(a) shows the frequencies of answers to the question: *How often do you apply software patterns in your software projects?* Possible answers were *Never*, *In less than 25% of the projects*, *In at least 25% of the projects*, *In every project*. Additionally to the frequency of answers, median values are shown. They were determined by assigning each of the answers to a single value in an ordinal scale ranging from zero to three and calculating the medians based on these numbers. While all other values are aligned to the left vertical axis, the medians are aligned to the right vertical axis. The figures show that the median for the control group is higher than for the pattern group. This means that the participants had more pattern and recovery experience than the participants in the pattern group.

The frequencies of answers to the question: *How many software patterns do you know well?* are shown in Fig. 3(b). Possible answers were *Less than 5 patterns*, *Six to ten patterns*, *Eleven to twenty patterns* and *More than 20 patterns*.

The participants were also asked whether they did architectural recovery before. The results are shown in Fig. 3(c).

The next two questions from the second questionnaire concerned the usage and helpfulness of patterns and show a small delta between the groups. First, the participants were asked to rate the helpfulness of patterns during architecture recovery on a scale from one for *not helpful* to five for *very helpful*. The median in both groups is four. These results are subjective in nature, as they express opinions. However, it shows that the members of both groups generally consider patterns as very useful in architecture recovery.

In the next question the participants were asked to estimate how extensively they used patterns during the recovery. Possible answers ranged from one for *almost never* to five for *very often*. The median answer in the pattern group is three, the median answer in the control group is two.

We looked into the types of the recovered decisions to see if the subjective estimations of the participants reflect the reality or not. In particular, the decisions were classified into pattern-related decisions (if the name of a pattern is literally mentioned in the decision section of the templates that were used to document decisions) and others (i.e. non-pattern-related). The results of this analysis can be found in Table 9 in the appendix.

The average number of pattern-related decisions per participant in the pattern group is 4.6 compared to 1.88 in the control group. The average number of other decisions per participant in the pattern group is 2.07 compared to 4.38 in the control group. The ratio of the pattern-related type to the other type is 2.23 in the pattern group compared to 0.43 in the control group. This shows that the members of the pattern group clearly focused more on patterns than the members of the control group. Independently from the group in which decisions were taken, the median quality of pattern-related decisions is 2.5, the median quality for other decisions is 2. This analysis of decision types has two results. It verifies that the pattern group focused on identifying pattern-related decisions and it shows that the difference in quality of decisions presented above can be ascribed to the focus on patterns.

Finally, we asked the participants to briefly describe how the recovery was performed. This was primarily done to confirm that the pattern group followed a pattern-based approach and to find out if the control group used any other systematic way to identify and describe decisions. Although the amount of qualitative data for this question was low (roughly one sentence per participant), we use the constant comparative method as originally described by Glaser and Strauss [11] to systematize the analysis of the answers. Therefore, we grouped (partial) answers to the question how the recovery was performed (*incidents* in the terminology of Glaser and Strauss) into categories (*coding*). Each answer was compared to the previously coded answers in the same and other categories to gain a better understanding of the decision recovery process they describe. Finally, the categories elicited from the control group were compared to the categories from the pattern group.

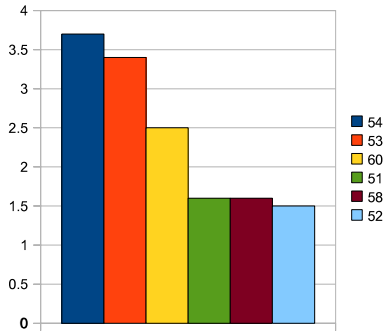
The results imply that the control group followed an intuitive approach, which was mainly driven by personal experience. Four participants answered that they searched for buzzwords that would remind them of a familiar technical solution. Three respondents stated that they read the textual descriptions in the architecture document to mine decisions; two analyzed the given UML diagrams. Three participants from the control group explicitly answered that they searched for patterns in the architecture. The other answers were not assigned to a specific category. However, one of these answers extremely represents the impression we gained during the analysis of the answers of the control group: "It looks like decision -> it is decision".

The answers of the pattern group reflect the focus on patterns. Thirteen out of 18 answers explicitly described an approach that centers on patterns. Six participants answered that they searched the UML diagrams for potential pattern participants. Four respondents identified candidate pattern decisions in the architecture documentation and then read up on the pattern in the pattern catalog, before they documented the decision using the given template.

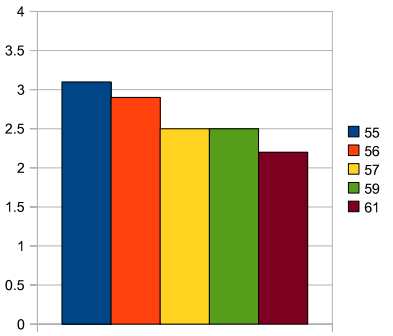
5.2. Dataset reduction

Outliers are potential candidates for dataset reduction, i.e. data points that are either much higher, or much lower than other data points. To find potential outliers, we calculated the average quality of decisions for each participant. Fig. 4 shows bar charts for every member of the control and the pattern groups for both executions of the experiment. The first two figures represent the participants at EuroPLoP 2009, the latter two represent the participants from the software architecture workshop in 2011. The numbers in the legends are the participant numbers.

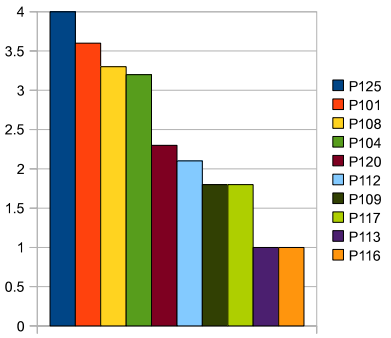
It is noticeable that two participants from the control group at EuroPLoP (Fig. 4(a) reached a significantly higher quality than the other members of this group. A closer analysis showed that most of their decisions concerned patterns. Three out of five decisions from participant 54 were pattern decisions; six out of eight decisions from participant 53 were pattern decisions. This could lead to the conclusion that the high decision quality of these two participants results from the focus



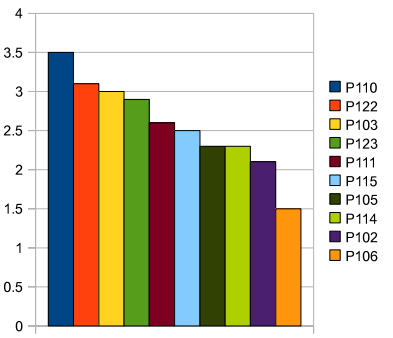
(a) Avg quality EuroPLoP Control Group.



(b) Avg quality EuroPLoP Pattern Group.



(c) Avg quality SWA WS Control Group.



(d) Avg quality SWA WS Pattern Group.

Fig. 4. Average quality per participant.

Table 7
Independent *t*-test for quality of decisions.

Factor	Mean diff.	*t*-value	*p*-value
Control group vs. pattern group	−0.4261	−3.222	0.001

Table 8
Independent *t*-test for quantity of decisions.

Factor	Mean diff.	*t*-value	*p*-value
Pattern group vs. control group	−0.250	−0.177	0.861

on patterns. However, their decisions were not excluded as outliers, because the difference to the other participants is not strong enough. Excluding the data points would have introduced a potential vulnerability of the study results.

5.3. Hypothesis testing

The two hypotheses regarding higher quality and quantity of recovered decisions when architecture recovery is focused on identifying patterns are evaluated using *t*-tests.

5.3.1. Quality of decisions

The results from the *t*-test (unpaired, two-tailed) are shown in Table 7. It provides strong evidence that H_{01} can be rejected. There is a noticeable difference in the quality of the recovered decisions between the pattern group and the control group. The *p*-value is very low, so the results are highly significant. Even if the classification of the used Likert-scale for the quality ratings of the decisions as interval scale could not be accepted, the descriptive statistics would still strongly support the result of the *t*-test, as the median value and the frequency of measured quality both support a result in favor of the pattern group.

5.3.2. Quantity of decisions

Hypothesis H_{02} was also evaluated with a *t*-test (unpaired, two-tailed). The results are shown in Table 8. Although slight differences in terms of the mean values can be observed, we are unable to show that this result is significant. An exclusion of the cases that were rated as non-architectural by one of the analysts would not have had an impact on this result.

80909

Problem / Issue	Client calls instane method on server side
Decision	Use explicit invocation
Alternatives (optional)	Use implicit ~~xxxx~~ invocation for async calls.
Arguments	The client can work on with the recieved result and use it in further calculations
Assumptions/ Constraints (optional)	Fast server response Low latency
Related requirements	bidirectional (network) communication / Full Duplex
Related decisions (optional)	Explicit Invocation Pattern
Additional Comments (optional)	

Fig. 5. Example 1 for pattern type decisions.

6. Interpretation

6.1. Evaluation of results and implications

6.1.1. Quality of decisions

Hypotheses H_{01} and H_1 concern the quality of recovered decisions. As pointed out in Section 5, we are able to provide strong evidence that the null-hypothesis H_{01} can be rejected.

Thus, the quality of decisions gained during architecture recovery is higher if the recovery focuses on identifying applied patterns. Additionally to the generally higher quality, the variance in the pattern group is much lower than in the pattern group.

34730

Problem / Issue	Many different aspects need to be handled in one call.
Decision	Pipes & Filters (aspects)
Alternatives (optional)	
Arguments	Transparent looking in of individual aspects
Assumptions/ Constraints (optional)	
Related requirements	
Related decisions (optional)	
Additional Comments (optional)	

Fig. 6. Example 2 for pattern type decisions.

We interpret our findings as follows. Patterns provide rich information about their problem- and solution spaces as well as reasoning for applying them in a system. They contain a great part of the architectural knowledge that is relevant for the system in which they were applied. If a pattern was identified during the recovery process, then the pattern documentation or the personal knowledge about the pattern helps to recover the intent of the original architect who decided to apply it. Of course, it still takes some effort to identify the pattern and customize the pattern's documented knowledge for the system at hand; but a large part of that high-quality knowledge is reused, not invented. The fact that the variance in the pattern group is relatively low shows that patterns help to reduce the dependency on individual abilities of the person doing the recovery. A certain quality level can be achieved even by people who do not have a strong background in architecture recovery. In contrast to this, the higher variance in the control group might stem from the different abilities of the participants.

6.1.2. Quantity of decisions

Hypotheses H_{02} and H_2 concern the quantity of recovered decisions. The results do not provide evidence to confirm or reject the null-hypothesis H_{02}. We are unable to show that the focus on patterns in architecture recovery has a significant effect on the quantity of decisions, i.e. the number of recovered decisions. This result is surprising to us. As described in the introductory section, we assumed that the quantity of recovered decisions would be higher in the pattern group. The results

Your Participant Number **41417**

Problem / Issue	How should the EJB container create the implementations of EJBHome / EJBObject required by the Spec?
Decision	Use dynamic proxies to implement EJBHome and EJBObject.
Alternatives (optional)	Use a precompiler to generate the objects at deployment time.
Arguments	- code (proxy generation at runtime is less fragile than at deployment time - dynamic proxies are JDK standard mean
Assumptions/ Constraints (optional)	The construction of dynamic proxies via reflection is cheap compared to EJB deployment time
Related requirements	- improve system reliability - ensure minimized JBoss implementation effort by reusing standard JDK mean
Related decisions (optional)	
Additional Comments (optional)	

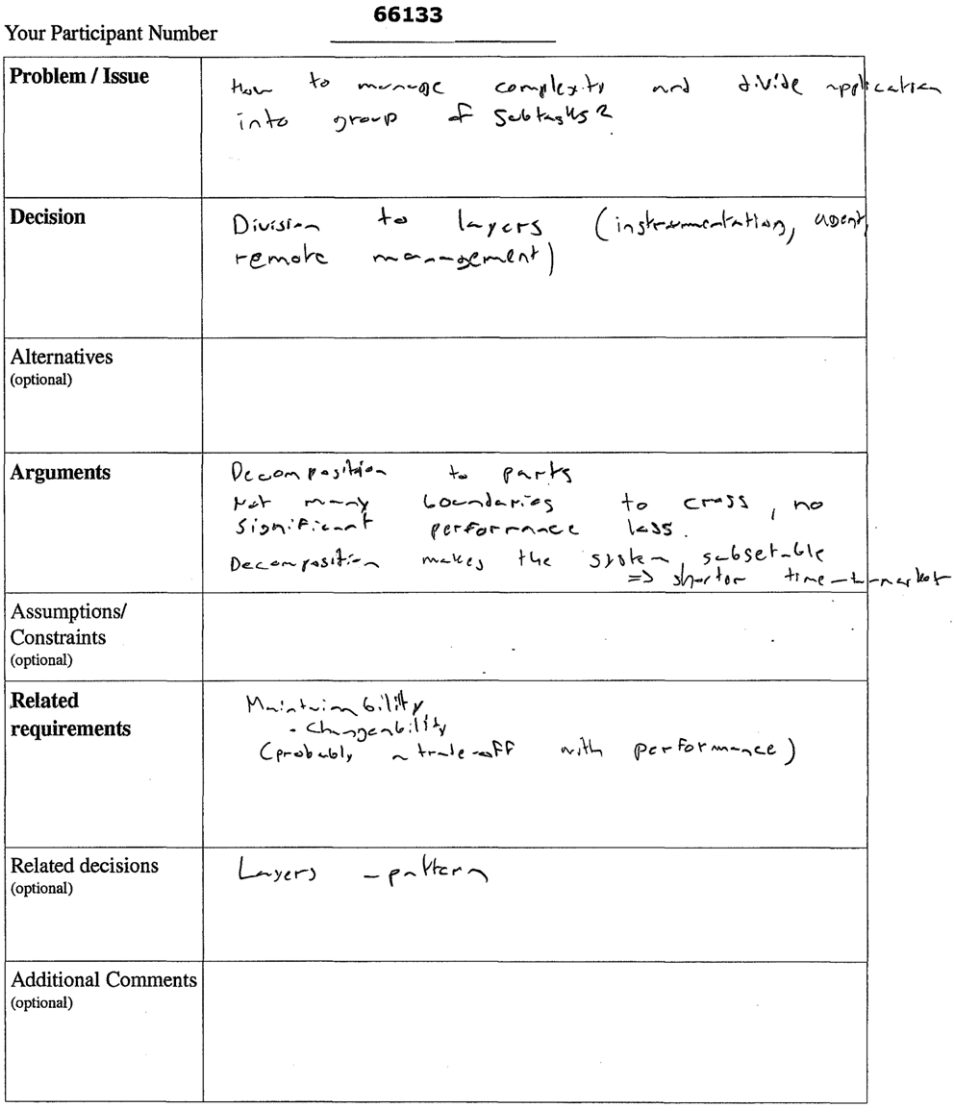
Fig. 7. Example 3 for pattern type decisions.

might stem from the fact that the participants in the pattern group took more time to document every single decision than the participants in the control group and thus had less time left to identify decisions. They also needed time to study the pattern catalog. This effect could possibly be eliminated by adjusting the study design. We will discuss this in Section 6.3.

It is interesting to note that the variance of the quantity was much higher in the control group than in the pattern group (19.267 compared to 11.286). This is another indicator for the lower dependency on the recoverer's personal skills and abilities as already discussed for the quality of decisions.

6.2. Limitations of the study

Several levels of validity have to be considered in this experiment. We consider the classification scheme for validity in experiments by Cook and Campbell [9]. Internal validity concerns the cause effect relationship between the treatment and the dependent variables measured in an experiment. External validity focuses on the generalizability of the results for a larger population. Conclusion validity focuses on the relationship between treatment and outcome and on the ability to draw conclusions from this relationship. Finally, construct validity is about the suitability of the study design for the theory behind the experiment. All threats to validity are categorized according to this classification.



Your Participant Number 66133

Problem / Issue	how to manage complexity and divide application into group of subtasks?
Decision	Division to layers (instrumentation, agent, remote management)
Alternatives (optional)	
Arguments	Decomposition to parts Not many boundaries to cross, no significant performance loss. Decomposition makes the system subsetable => shorter time-to-market
Assumptions/ Constraints (optional)	
Related requirements	Maintainability - Changeability (probably ~ trade-off with performance)
Related decisions (optional)	Layers - pattern
Additional Comments (optional)	

Fig. 8. Example 4 for pattern type decisions.

6.2.1. Internal validity

- The object in the experiment was a documentation of an object-oriented middleware. In this particular case, the JBoss application server, many architectural patterns were implicitly and explicitly applied in the system, which might lead to the conclusion that the pattern group had advantages compared to the control group. This, however, does not seem to be the case. Both groups could have identified the architecture decisions behind the applied patterns. Also many other architecture decisions were made by the original architects that do not concern patterns, e.g. the choice of used frameworks or programming libraries. Finally, although many patterns were applied in the JBoss server, our results do not confirm that a focus on patterns leads to higher quantity of decisions. Thus, the fact that the JBoss design contains a lot patterns did not have an effect in our study.

  Another potential threat related to the choice of JBoss as object of the study is the fact that many J2EE patterns exists. The former SUN catalog of J2EE patterns is one source of such patterns [32]. However, the J2EE patterns support the creation of applications that conform to the J2EE specification set. To the best of our knowledge, no pattern catalog or pattern language exists that is specific to developing J2EE servers. In this case, the application analyzed by the participants was a J2EE server, not a J2EE application. Thus, we do not consider this a threat to validity.

  We conclude that the choice of the object studied in this experiment is not a threat to the internal validity of the results.
- The outcome of the experiment could have been different for systems, in which fewer or no patterns were applied by the designers. Normally, it is more difficult to identify pattern decisions in systems, in which not many patterns have been applied. However, in a study of pattern usage that we conducted [13], we found that most systems have at

Your Participant Number **51204**

Problem / Issue	The architect needs to decide on how to implement a micro kernel system
Decision	Use JMX as micro kernel architecture.
Alternatives (optional)	- implement custom micro kernel on your own - use some other framework, e.g. OSGI
Arguments	- JMX is a standardized Java technology, allowing for simple 3rd party integration - JMX exists already, lowering cost - JMX offers monitoring facilities
Assumptions/ Constraints (optional)	Architecture based on micro kernel
Related requirements	- Base development on standard technologies - Allow for runtime monitoring
Related decisions (optional)	Use micro kernel architecture. (#1) Use 100% Java (#5)
Additional Comments (optional)	

Fig. 9. Example 5 for pattern type decisions.

least two architecture patterns, some have as many as eight. Furthermore, besides architecture patterns, the pattern community has assembled a vast body of pattern knowledge for virtually all software domains; thus several patterns can be potentially found in any system. Moreover, even if patterns are not consciously used by designers, they can still be applied unconsciously, as designers tend to reach common solutions. It is of course unlikely that all architecture decisions in a system are pattern related, but even in cases where only a few patterns were used, the decisions can be an important entry point for the recovery of the remaining decisions, because decisions are usually interrelated. The threat, however, cannot be mitigated completely.

In a few rare cases, so many patterns could have been applied in an architecture that individual patterns are hard to identify in the design. This, however, is a theoretical problem that is not very likely to be observed in reality. We do no consider it a threat to validity.

- Typically, there is a variation in human performance that might influence the results of the experiments. This can distort the results, because then the performance would not arise from the difference in treatments. We tried to minimize this factor by balancing the two groups concerning the relevant previous experience of the participants. The groups were well balanced in all categories, namely programming experience, middleware experience, architecture experience and recovery experience. Thus, this factor is not seen as a threat to validity.
- The control group could theoretically have imitated the behavior of the pattern group. In this particular experiment, the two groups performed in two different rooms at the same time. The instructions that concerned the difference in

Your Participant Number 68678

Problem / Issue	Which programming environment should be used for the JBoss implementation?
Decision	Implement JBoss on 100% pure Java
Alternatives (optional)	- integrate platform-specific code via JNI
Arguments	- pure JAVA runs on any JVM - only single platform code is compiled to - reduced testing effort for multiple platforms
Assumptions/ Constraints (optional)	- performance required can be done in 100% Java - all interfaces to required external resources (Network, database) are accessible via Java API
Related requirements	- support multiple target platforms at minimal cost - reduce/unify required skill set of JBoss developers
Related decisions (optional)	Use JVM as micro kernel (#2)
Additional Comments (optional)	Might lead to performance problems; Testing multiple JVMs / Platforms is facilitated, but not unnecessary

Fig. 10. Example 1 for other type decisions.

treatments were given to the participants after they moved into these rooms. That way, there was no chance for the control group to consciously or unconsciously imitate the behavior of the pattern group.

- The raters could have unconsciously ranked the pattern decisions higher than other decisions, because patterns contain professionally edited material that is succinct and easy to comprehend. The data gathered in both executions, however, shows that the participants used the patterns to interpret the architectural solutions found in the JBoss architecture and documented the decisions using their own words, adapting the pattern information in the context of the JBoss system. Therefore, decisions by and large, were not documented by copying or reusing the text from the pattern catalogs.

6.2.2. External validity

- The subject population in the experiment might not be representative for a larger population. In this case, the subjects (participants) of the first execution of the experiment were participants of the EuroPLoP conference. They all have an academic or industrial background in several software engineering disciplines and a strong interest in patterns. The second execution at the software architecture workshop in Venlo was conducted mainly with industrial practitioners from different domains.

  Our results imply that the affiliation does not have an influence on the external validity of results. No correlation between the affiliation and the quality of recovered decision could be found. Additionally, each of the two executions analyzed in isolation would have lead to the same conclusions, namely that a focus on patterns leads to higher quality, but not to higher quantity. Therefore, we conclude that the pattern background of the EuroPLoP participants does not distort the study results.

Your Participant Number 15787

Problem / Issue	The Architect needed to provide access to MBean Server to (heterogeneous) remote Applications
Decision	Provide Access via Connectors & Adaptors
Alternatives (optional)	Define standard protocol & API for Accessing the MBean Server
Arguments	Using connectors & Adapters Allows Any remote Application to Access the MBean Server, As long As that Application supports one of communication frameworks (RMI, ...) & protocol (SNMP, ...). IF a standard protocol is Assumed, each Application must be re-designed to support interaction with MBean Server
Assumptions/ Constraints (optional)	For Each new communication framework / protocol A new connector / Adaptor must be provided.
Related requirements	• improves Scalability / not sure About the term. I mean, you can Add other Application that want to Access the server. "Extensibility"? • Re-usability? (each time you Add new connector/Adaptor it can be re-used next time for other Applications using the same framework/protocol)
Related decisions (optional)	
Additional Comments (optional)	not All frameworks / protocols can be good for communication with the server.

Fig. 11. Example 2 for other type decisions.

- The instrumentation and object in the experiment might have been unrealistic or old-fashioned. In this case, the architecture recovery was based on a printed architecture documentation. Usually different tools would be used to support architecture recovery. Code analyzers, reverse engineering tools and dependency analysis tools are some examples. These tools are primarily used to recover the design of a software system. In this experiment, for practical reasons, the design of the software was readily provided in a printed document. The focus was on architecture decision recovery, not on architecture design recovery. We assume that the measured effect of a pattern focus during architecture decision recovery is independent from the way, in which the design was recovered.

  Another theoretical thread to validity is that the problem in the analysis might be unrealistic and too simple to allow generalization. This was not the case here. The object used is an excerpt from a real documentation of the JBoss server that was not created for the purpose of this experiment.
- Finally, the experimenters could have biased the measurements of the independent variables. We mitigated this risk by assigning the quality ratings of the decisions to two independent experts that had no knowledge about the goals of the experiment. Additionally, by using pseudonymization, the analysts had no chance to guess which decisions belonged to which group. They could not even have found out which decisions belonged together (were documented by the same participant).

6.2.3. Conclusion validity

- As discussed in the design section, there is a potential threat to validity resulting from the interpretation of the Likert-scale, which was used to rate the quality of architecture decisions, as an interval scale. Some of the statistical tests used to analyze the results (mean, variance, standard deviation and t-test) would not have been valid for nominal scale types.

Your Participant Number **30924**

Problem / Issue	HOW TO PROVIDE ACCESS TO THE JBOSS INFRASTRUCTURE FOR REMOTE COMPONENTS, EACH USING A DIFFERENT REMOTING / PROTOCOL TECHNOLOGY ?. IN ADDITION, HOW CAN ONE KEEP THE NUMBER & KIND OF CONNECTORS EXTENSIBLE?
Decision	USING A KIND OF SERVICE ABSTRACTION ON TOP OF THE MBEAN SERVER
Alternatives (optional)	• STIPULATE A SINGLE REMOTING STRATEGY / CHANNEL, REMOTE ENDS HAVE TO COMPLY WITH •
Arguments	• INCREASE REACH & INTEGRATABILITY IN HETEROGENEOUS SYSTEM LANDSCAPES → e.g. in in corporate context • REDUCE POTENTIAL ENTRY / ADAPTATION COSTS PER CUSTOMER → AVOID ON-SITE INTEGRATION EFFORTS • REDUCE DEVELOPMENT EFFORT TO FREESTANDING EXTENSIONS
Assumptions/ Constraints (optional)	• TARGETED SYSTEM LANDSCAPE IS BASED ON ESTABLISHED MIDDLEWARE STANDARDS (NO AD-HOC OR PROPRIETARY SOLUTIONS)
Related requirements	• INCREASED INTEGRATABILITY (LEGACY INTEGRATION) • COPING WITH SYSTEM HETEROGENEITY • AVOID LOCK-INS ; EASEN FURTHER-DEVELOPMENT OF THE OVERALL INFRASTRUCTURE (i.e., allow for TECHNOLOGY CHANGE IN REMOTING)
Related decisions (optional)	• SEE PAGE ② : "LAYERING OF MESSAGE INFRASTRUCTURE" (JMX)
Additional Comments (optional)	

IF A NEW TECHNOLOGY ENERGES // IS REQUIRED

Fig. 12. Example 3 for other type decisions.

We argue that in this particular situation the ratings of the Likert-scale are metrically scaled, and thus have the character of an interval scale. This means for instance that the quality rating four is actually two-times higher than the quality rating two. Because this interpretation remains critical, we also calculate the median for the quality ratings, which would also be applicable for nominal scales.

- Another potential threat to validity is the subjectivity of the scale used to rate the quality. We tried to mitigate this risk by asking two independent experts in the field of software architecture to rate the quality of every recovered decision. In the analysis, we took the arithmetic average of the two ratings per decision as a basis. However, the null-hypothesis would also have been rejected for the results of both analysts individually. Additionally, from the fact that our result has a very high significance, we conclude that this potential threat is mitigated.

6.2.4. Construct validity

- The fact that only one object; the JBoss documentation; was used in the experiment, introduces the risk that the cause construct is underrepresented. Theoretically, the results could look different if multiple architecture documentations would be used for the recovery. We assume that the used system and its documentation are representative for large and medium-size object-oriented systems. The threat, however, cannot totally be ignored.
- Another potential threat to validity is the number of measures used to evaluate the quality of recovered decisions. In our case we only used one variable to measure the quality of the recovered decisions. This does not allow cross-checking the results with different measures.

37389

Problem / Issue	Which license model should we use? This has consequences for which ~~external~~ existing SW we can incorporate
Decision	LGPL (open source)
Alternatives (optional)	proprietary < free / commercial GPL ⋮
Arguments	With open source our system will ~~probably~~ be used more than proprietary. Moreover, we will get source-level feedback from users. GPL would restrict our users too much.
Assumptions/ Constraints (optional)	We have a business model that ensures our continued existence without having to charge for our software
Related requirements	~~-~~ – SW should be free – source should be available for easy debugging – can use ~~Java~~ SW without opening up all my software
Related decisions (optional)	– use Java (depend on Java only) – Follow J2EE spec (and many others)
Additional Comments (optional)	there are a lot of such decisions already in the introduction

Fig. 13. Example 4 for other type decisions.

6.3. Lessons learned

The analysis of the quantity of decisions showed that in average the control group recovered more decisions than the pattern group. We already presumed that one of the reasons for this outcome might be that the participants in the pattern group took more time to document every single decision than the participants in the control group and therefore had less time left to identify decisions. Besides, they took time to study the pattern material. The latter was particularly the case during the workshop in Venlo. The participants at EuroPLoP had presumably more knowledge about patterns and consequently took less time to study the pattern material.

One way of eliminating this effect would have been to assign more time to the pattern group than the control group. But as the additional time needed to document decisions can hardly be estimated or even predicted, it would have been hard to define an adequate period of time to add it to the pattern group's experiment run-time. Additionally, a potential threat to validity would have been introduced.

Another possible improvement of the study design concerns the data collection. In this experiment, we could not make use of computers or other electronic devices to collect data. This was a handicap during the analysis. Gathering data electronically using online surveys and electronic forms would have eased the analysis. However, participants felt comfortable with the architecture documentation on paper, because the paper form allowed them to take notes.

7. Conclusions and future work

In this paper, we describe the results of a controlled experiment that was conducted to find out if patterns are beneficial for architecture decision recovery. Two aspects were specifically taken into consideration: the quality and the quantity of

11958

Problem / Issue	How to keep system development and understanding manageable?
Decision	Subdivide into modules (EJB container, JBossTx, ...)
Alternatives (optional)	– use a thick soup of classes – use a different module subdivision
Arguments	A subdivision into modules helps to distribute development effort and allows incremental learning of the modules.
Assumptions/ Constraints (optional)	
Related requirements	- system should be easy to learn for users – development effort should stay within reasonable limits
Related decisions (optional)	
Additional Comments (optional)	

Fig. 14. Example 5 for other type decisions.

recovered decisions. The evaluation of the experiment shows that a focus on patterns leads to significantly higher and stable quality of decisions, compared to intuitive recovery, which leads to a lower quality with higher variance. We are unable to show that the quantity of recovered decisions is also positively affected.

In the future, we plan to replicate the experiment with different types of software systems from other application domains, which are less pattern-intensive than the object used in this study.

Another direction for future work is to find out if besides patterns, there are other forms of generic architectural knowledge that can be beneficial in architecture decision recovery. In the context of a research project, we developed a publicly available online repository for patterns and technologies [35]. The basis for the repository is a common meta model for patterns and technologies that allows to relate patterns, pattern variants and software technologies. We plan to use the tool for a follow up experiment, in which we allow the treatment group to use all kinds of generic architectural knowledge, instead of focusing on patterns.

Acknowledgements

We would like to thank Anton Jansen and Chuck Allison for analyzing the results of the study. We also thank the members of the EuroPLoP 2009 focus group and the participants of the software architecture workshop in Venlo 2011 for taking part in the experiment.

Appendix

Raw data—quality ratings and decision types

See Table 9.

Table 9
Quality ratings and decision types.

Execution Group	Group	Participant	Decision	Decision Type	Analyst 1	Analyst 2	Average
EuroPLoP	Control	51	15639	Pattern-Type	3	1	2
EuroPLoP	Control	51	31219	Other	1	2	1.5
EuroPLoP	Control	51	35295	Other	2	1	1.5
EuroPLoP	Control	51	57732	Other	1	1	1
EuroPLoP	Control	51	58823	Pattern-Type	3	1	2
EuroPLoP	Control	52	19704	Other	1	1	1
EuroPLoP	Control	52	21539	Other	1	1	1
EuroPLoP	Control	52	23027	Other	1	1	1
EuroPLoP	Control	52	24025	Other	3	1	2
EuroPLoP	Control	52	28014	Other	3	2	2.5
EuroPLoP	Control	52	29573	Other	1	1	1
EuroPLoP	Control	52	30292	Other	2	1	1.5
EuroPLoP	Control	52	33148	Other	3	1	2
EuroPLoP	Control	52	41748	Other	1	1	1
EuroPLoP	Control	52	42404	Other	2	1	1.5
EuroPLoP	Control	52	45167	Other	3	1	2
EuroPLoP	Control	52	51001	Other	2	1	1.5
EuroPLoP	Control	52	51331	Pattern-Type	2	1	1.5
EuroPLoP	Control	52	55931	Other	3	1	2
EuroPLoP	Control	52	56835	Other	3	1	2
EuroPLoP	Control	52	63345	Other	1	1	1
EuroPLoP	Control	52	63653	Other	1	2	1.5
EuroPLoP	Control	52	69428	Pattern-Type	2	1	1.5
EuroPLoP	Control	52	76435	Other	2	1	1.5
EuroPLoP	Control	53	15154	Pattern-Type	5	3	4
EuroPLoP	Control	53	23289	Pattern-Type	4	2	3
EuroPLoP	Control	53	25919	Pattern-Type	5	2	3.5
EuroPLoP	Control	53	35942	Other	5	2	3.5
EuroPLoP	Control	53	41417	Pattern-Type	5	2	3.5
EuroPLoP	Control	53	46622	Pattern-Type	4	3	3.5
EuroPLoP	Control	53	51204	Pattern-Type	5	2	3.5
EuroPLoP	Control	53	68678	Other	4	2	3
EuroPLoP	Control	54	30924	Other	4	3	3.5
EuroPLoP	Control	54	41044	Other	4	3	3.5
EuroPLoP	Control	54	52975	Pattern-Type	4	3	3.5
EuroPLoP	Control	54	53685	Pattern-Type	5	2	3.5
EuroPLoP	Control	54	64601	Pattern-Type	5	4	4.5
EuroPLoP	Control	58	13969	Other	2	1	1.5
EuroPLoP	Control	58	14458	Other	2	2	2
EuroPLoP	Control	58	18775	Other	3	1	2
EuroPLoP	Control	58	21221	Other	2	1	1.5
EuroPLoP	Control	58	26497	Other	1	2	1.5
EuroPLoP	Control	58	26994	Pattern-Type	3	1	2
EuroPLoP	Control	58	34902	Pattern-Type	2	1	1.5
EuroPLoP	Control	58	35617	Other	1	1	1
EuroPLoP	Control	58	44214	Other	1	1	1
EuroPLoP	Control	58	44467	Other	1	2	1.5
EuroPLoP	Control	58	47099	Other	2	2	2
EuroPLoP	Control	58	54821	Pattern-Type	2	2	2
EuroPLoP	Control	58	73170	Other	2	1	1.5
EuroPLoP	Control	60	15787	Other	4	3	3.5

(Continued on next page)

Execution Group	Group	Participant	Decision	Decision Type	Analyst 1	Analyst 2	Average
EuroPLoP	Control	60	25333	Other	2	2	2
EuroPLoP	Control	60	33726	Other	3	1	2
EuroPLoP	Control	60	33899	Other	4	1	2.5
EuroPLoP	Control	60	53821	Other	4	1	2.5
EuroPLoP	Pattern	55	21906	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	55	26522	Other	4	3	3.5
EuroPLoP	Pattern	55	26637	Other	4	2	3
EuroPLoP	Pattern	55	29399	Other	4	2	3
EuroPLoP	Pattern	55	35177	Other	4	2	3
EuroPLoP	Pattern	55	42037	Other	4	2	3
EuroPLoP	Pattern	55	58456	Other	3	2	2.5
EuroPLoP	Pattern	55	66133	Pattern-Type	4	3	3.5
EuroPLoP	Pattern	55	77736	Other	5	2	3.5
EuroPLoP	Pattern	56	19520	Other	5	2	3.5
EuroPLoP	Pattern	56	29339	Other	4	1	2.5
EuroPLoP	Pattern	56	31499	Other	4	3	3.5
EuroPLoP	Pattern	56	44253	Pattern-Type	4	3	3.5
EuroPLoP	Pattern	56	46178	Pattern-Type	3	3	3
EuroPLoP	Pattern	56	68057	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	56	74785	Other	2	2	2
EuroPLoP	Pattern	57	14562	Pattern-Type	1	2	1.5
EuroPLoP	Pattern	57	21778	Pattern-Type	4	1	2.5
EuroPLoP	Pattern	57	25077	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	57	31053	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	57	36371	Pattern-Type	4	2	3
EuroPLoP	Pattern	57	47861	Pattern-Type	4	2	3
EuroPLoP	Pattern	57	49896	Other	4	2	3
EuroPLoP	Pattern	57	58724	Pattern-Type	4	2	3
EuroPLoP	Pattern	57	59375	Pattern-Type	3	1	2
EuroPLoP	Pattern	57	64255	Pattern-Type	2	2	2
EuroPLoP	Pattern	57	70273	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	57	71108	Pattern-Type	4	1	2.5
EuroPLoP	Pattern	57	71172	Pattern-Type	4	2	3
EuroPLoP	Pattern	57	72293	Other	3	2	2.5
EuroPLoP	Pattern	59	14001	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	59	16418	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	59	16548	Pattern-Type	4	1	2.5
EuroPLoP	Pattern	59	18326	Pattern-Type	4	3	3.5
EuroPLoP	Pattern	59	19641	Pattern-Type	2	2	2
EuroPLoP	Pattern	59	27679	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	59	34381	Pattern-Type	4	2	3
EuroPLoP	Pattern	59	36854	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	59	51205	Pattern-Type	5	2	3.5
EuroPLoP	Pattern	59	70245	Pattern-Type	3	1	2
EuroPLoP	Pattern	59	76602	Pattern-Type	1	2	1.5
EuroPLoP	Pattern	61	22955	Pattern-Type	3	1	2
EuroPLoP	Pattern	61	50589	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	61	50699	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	61	60055	Pattern-Type	2	2	2
EuroPLoP	Pattern	61	62783	Pattern-Type	3	2	2.5
EuroPLoP	Pattern	61	69068	Pattern-Type	2	1	1.5
SWA Workshop	Control	P101	11958	Other	5	3	4
SWA Workshop	Control	P101	20924	Other	4	1	2.5
SWA Workshop	Control	P101	37389	Other	5	5	5
SWA Workshop	Control	P101	46596	Other	4	2	3
SWA Workshop	Control	P101	70401	Pattern-Type	4	3	3.5
SWA Workshop	Control	P104	17130	Other	4	4	4
SWA Workshop	Control	P104	53026	Other	4	2	3

(Continued on next page)

Execution Group	Group	Participant	Decision	Decision Type	Analyst 1	Analyst 2	Average
SWA Workshop	Control	P104	61606	Other	4	1	2.5
SWA Workshop	Control	P104	73623	Pattern-Type	5	2	3.5
SWA Workshop	Control	P104	95115	Pattern-Type	3	3	3
SWA Workshop	Control	P108	26170	Other	5	2	3.5
SWA Workshop	Control	P108	34797	Other	3	1	2
SWA Workshop	Control	P108	48667	Other	4	2	3
SWA Workshop	Control	P108	53246	Other	4	2	3
SWA Workshop	Control	P108	80735	Other	5	5	5
SWA Workshop	Control	P108	88971	Pattern-Type	3	3	3
SWA Workshop	Control	P109	23445	Other	2	2	2
SWA Workshop	Control	P109	33167	Other	3	1	2
SWA Workshop	Control	P109	38131	Other	2	1	1.5
SWA Workshop	Control	P109	48170	Pattern-Type	2	2	2
SWA Workshop	Control	P109	79232	Pattern-Type	1	1	1
SWA Workshop	Control	P109	81928	Other	2	1	1.5
SWA Workshop	Control	P109	87890	Other	3	3	3
SWA Workshop	Control	P109	93340	Other	1	1	1
SWA Workshop	Control	P112	34730	Pattern-Type	2	2	2
SWA Workshop	Control	P112	57609	Pattern-Type	3	1	2
SWA Workshop	Control	P112	59805	Other	2	2	2
SWA Workshop	Control	P112	68209	Other	4	2	3
SWA Workshop	Control	P112	76889	Pattern-Type	3	1	2
SWA Workshop	Control	P112	97764	Pattern-Type	2	1	1.5
SWA Workshop	Control	P113	38562	Other	1	1	1
SWA Workshop	Control	P113	51690	Pattern-Type	1	1	1
SWA Workshop	Control	P113	53126	Other	1	1	1
SWA Workshop	Control	P113	70648	Other	1	1	1
SWA Workshop	Control	P113	81229	Pattern-Type	1	1	1
SWA Workshop	Control	P116	33889	Other	1	1	1
SWA Workshop	Control	P116	53663	Other	1	1	1
SWA Workshop	Control	P116	64769	Other	1	1	1
SWA Workshop	Control	P116	71283	Pattern-Type	1	1	1
SWA Workshop	Control	P116	80071	Other	1	1	1
SWA Workshop	Control	P117	66942	Pattern-Type	1	1	1
SWA Workshop	Control	P117	81042	Other	3	2	2.5
SWA Workshop	Control	P120	32371	Other	4	2	3
SWA Workshop	Control	P120	49144	Other	2	1	1.5
SWA Workshop	Control	P125	25968	Other	5	3	4
SWA Workshop	Pattern	P102	14837	Pattern-Type	4	2	3
SWA Workshop	Pattern	P102	47817	Pattern-Type	2	1	1.5
SWA Workshop	Pattern	P102	75306	Other	2	2	2
SWA Workshop	Pattern	P102	92796	Other	3	1	2
SWA Workshop	Pattern	P103	21416	Pattern-Type	4	3	3.5
SWA Workshop	Pattern	P103	26721	Pattern-Type	4	3	3.5
SWA Workshop	Pattern	P103	53077	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P103	79679	Other	3	2	2.5
SWA Workshop	Pattern	P105	34596	Pattern-Type	3	1	2
SWA Workshop	Pattern	P105	73625	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P105	79704	Pattern-Type	4	1	2.5
SWA Workshop	Pattern	P106	31527	Pattern-Type	1	1	1
SWA Workshop	Pattern	P106	46146	Pattern-Type	2	2	2
SWA Workshop	Pattern	P106	65149	Pattern-Type	2	1	1.5
SWA Workshop	Pattern	P106	75358	Pattern-Type	2	1	1.5
SWA Workshop	Pattern	P110	37008	Other	3	3	3
SWA Workshop	Pattern	P110	61530	Other	5	3	4
SWA Workshop	Pattern	P111	14680	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P111	19077	Pattern-Type	4	3	3.5
SWA Workshop	Pattern	P111	27818	Pattern-Type	3	3	3

(Continued on next page)

Execution Group	Group	Participant	Decision	Decision Type	Analyst 1	Analyst 2	Average
SWA Workshop	Pattern	P111	33776	Other	2	2	2
SWA Workshop	Pattern	P111	44350	Pattern-Type	2	1	1.5
SWA Workshop	Pattern	P111	67646	Other	5	5	5
SWA Workshop	Pattern	P111	86855	Pattern-Type	2	1	1.5
SWA Workshop	Pattern	P111	90696	Pattern-Type	2	1	1.5
SWA Workshop	Pattern	P114	23961	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P114	41389	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P114	59052	Pattern-Type	4	2	3
SWA Workshop	Pattern	P114	76798	Other	1	1	1
SWA Workshop	Pattern	P115	41047	Pattern-Type	4	1	2.5
SWA Workshop	Pattern	P115	57170	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P115	70149	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P122	14967	Pattern-Type	2	1	1.5
SWA Workshop	Pattern	P122	56783	Pattern-Type	5	4	4.5
SWA Workshop	Pattern	P122	80909	Pattern-Type	4	2	3
SWA Workshop	Pattern	P122	90464	Pattern-Type	5	2	3.5
SWA Workshop	Pattern	P123	10498	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P123	16680	Pattern-Type	5	3	4
SWA Workshop	Pattern	P123	22025	Pattern-Type	4	3	3.5
SWA Workshop	Pattern	P123	23757	Pattern-Type	4	3	3.5
SWA Workshop	Pattern	P123	57463	Pattern-Type	3	2	2.5
SWA Workshop	Pattern	P123	66453	Pattern-Type	4	2	3
SWA Workshop	Pattern	P123	83213	Pattern-Type	2	1	1.5

Typical decisions recovered by the participants

In the following, we present typical examples of decisions recovered by the participants. The first five decisions are pattern decisions, the latter five are of other types (See Figs. 5–14.).

References

[1] C. Alexander, The Timeless way of Building, Oxford University Press, USA, 1979.
[2] P. Avgeriou, U. Zdun, Architectural patterns revisited–a pattern language, in: Proceedings of the 10th European Conference on Pattern Languages of Programs, EuroPlop, Irsee, 2005.
[3] M. 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, IEEE Computer Society, 2007, p. 11.
[4] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice, second edition, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
[5] B. Boehm, H. Rombach, M. Zelkowitz, Foundations of Empirical Software Engineering: The Legacy of Victor R. Basili, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
[6] J. Bosch, Software architecture: the next step, in: Lecture Notes in Computer Science, 2004, pp. 194–199.
[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture, Volume 1: A System of Patterns, John Wiley & Sons, Inc., New York, NY, USA, 1996.
[8] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, R. Little, Documenting Software Architectures: Views and Beyond, Pearson Education, 2002.
[9] T. Cook, D. Campbell, Quasi-Experimentation: Design and Analysis Issues for Field Settings, Houghton Mifflin, 1979.
[10] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, in: Addison-Wesley Professional Computing Series, 1995.
[11] B. Glaser, A. Strauss, The Discovery of Grounded Theory: Strategies for Qualitative Research, AldineTransaction, 1967.
[12] G. Goldstein, M. Hersen, Handbook of Psychological Assessment, Pergamon Press, 2000.
[13] N. Harrison, P. Avgeriou, Analysis of architecture pattern usage in legacy system architecture documentation, in: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture, WICSA 2008, IEEE Computer Society, 2008, pp. 147–156.
[14] N. Harrison, P. Avgeriou, U. Zdun, Using patterns to capture architectural decisions, IEEE software (2007) 38–45.
[15] Hillside Europe e.V., European Conference on Pattern Languages of Programs, http://hillside.net/europlop/, 2009.
[16] J.F. Hoorn, R. Farenhorst, P. Lago, H. van Vliet, The lonesome architect, Journal of Systems and Software 84 (9) (2011) 1424–1435.
[17] IEEE, IEEE Std 1471–2000, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, October 2000.
[18] ISO, Systems and software engineering — Architecture description, ISO/IEC/IEEE 42010, May 2011, pp. 1–46.
[19] A. Jansen, J. Bosch, Software architecture as a set of architectural design decisions, in: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society, 2005, pp. 109–120.
[20] A. Jansen, J. Bosch, P. Avgeriou, Documenting after the fact: recovering architectural design decisions, Journal of Systems and Software 81 (4) (2008) 536–557.
[21] JBoss.org., Community driven open source middleware, http://www.jboss.org/, Feb. 2011.
[22] A. Jedlitschka, D. Pfahl, Reporting guidelines for controlled experiments in software engineering, in: International Symposium on Empirical Software Engineering, IEEE, 2005, pp. 92–101.
[23] R. Kazman, S. Carrière, Playing detective: Reconstructing software architecture from available evidence, Automated Software Engineering 6 (2) (1999) 107–138.

[24] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, in: IEEE Transactions on Software Engineering, 2002, pp. 721–734.
[25] R. Koschke, Architecture reconstruction, in: Software Engineering, Springer-Verlag, 2009, pp. 140–173.
[26] R. Krikhaar, A. Postma, A. Sellink, M. Stroucken, C. Verhoef, A two-phase process for software architecture improvement, in: Proceedings of the IEEE International Conference on Software Maintenance, IEEE Computer Society, 1999, p. 371.
[27] P. Kruchten, The 4 + 1 view model of architecture, IEEE Software 12 (6) (1995) 42–50.
[28] J. Liu, Research Project: An Analysis of JBoss Architecture. http://www.huihoo.org/jboss/jboss.html, 2002.
[29] T. O'Gorman, Applied Adaptive Statistical Methods: Tests of Significance and Confidence Intervals, Society for Industrial Mathematics, 2004.
[30] D. Schmidt, F. Buschmann, Patterns, frameworks, and middleware: their synergistic relationships, in: Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society Washington, DC, USA, 2003, pp. 694–704.
[31] S. Stevens, On the theory of scales of measurement, Science 103 (2684) (1946) 677–680.
[32] Sun Microsystems, Core J2EE patterns, http://java.sun.com/blueprints/corej2eepatterns/, 2011.
[33] W. Trochim, J. Donnelly, The Research Methods Knowledge Base, Atomic Dog Publishing, Mason, OH, 2007.
[34] J. Tyree, A. Akerman, Architecture decisions: demystifying architecture, IEEE Software 22 (2) (2005) 19–27.
[35] University of Groningen, Software Engineering and Architecture Group, The Open Pattern Repository, http://code.google.com/p/openpatternrepository/, Feb. 2011.
[36] U. van Heesch, P. Avgeriou, A pattern driven approach against architectural knowledge vaporization, in: Proceedings of the 14th European Conference on Pattern Languages of Programs, EuroPLoP, Irsee, 2009.
[37] J. Ven, A. Jansen, J. Nijhuis, J. Bosch, Design Decisions: The Bridge between Rationale and Architecture, Springer, 2006, pp. 329– 348.
[38] C. Wohlin, M. Hoest, P. Runeson, M. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Pub., 2000.
[39] H. Yan, D. Garlan, B. Schmerl, J. Aldrich, R. Kazman, Discotect: a system for discovering architectures from running systems, in: Proceedings of the 26th International Conference on Software Engineering, IEEE Computer Society, 2004, p. 479.
[40] O. Zimmermann, J. Grundler, S. Tai, F. Leymann, Architectural decisions and patterns for transactional workflows in soa, in: Proceedings of the 5th International Conference on Service-Oriented Computing, Springer-Verlag, 2007, pp. 81–93.
[41] O. Zimmermann, U. Zdun, T. Gschwind, Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method, in: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture, WICSA 2008, IEEE Computer Society, 2008, pp. 157–166.