Design and Evaluation of a Process for Identifying Architecture Patterns in Open Source Software

Klaas-Jan Stol[1], Paris Avgeriou[2], and Muhammad Ali Babar[3]

[1] Lero—The Irish Software Engineering Research Centre, University of Limerick, Ireland
[2] University of Groningen, The Netherlands
[3] IT University of Copenhagen, Denmark
klaas-jan.stol@lero.ie, paris@cs.rug.nl, malibaba@itu.dk

Abstract. Architecture patterns have a direct effect (positive or negative) on a system's quality attributes (e.g., performance). Therefore, information about patterns used in a product can provide valuable insights to, e.g., component integrators who wish to evaluate a software product. Unfortunately, this information is often not readily available, in particular for Open Source Software (OSS) products, which are increasingly used in component-based development. This paper presents the design and evaluation of a process for Identifying Architecture Patterns in OSS ("IDAPO"). The results of the evaluation suggest that IDAPO is helpful to identify potentially present patterns, and that a process framework may provide better opportunities for tailoring to the users' needs.

Keywords: architecture patterns, quality attributes, open source software, empirical evaluation, quasi-experiment.

1 Introduction

Architecture patterns (e.g., layers, model-view-controller) are generalized solutions to recurring system-wide design problems, which describe the main roles of system partitions and their interactions. It is widely recognized that architecture patterns have a direct effect on a system's quality attributes (QAs), such as performance and reliability [1]. Integrating components into a system whose overall QAs are incompatible with the component's QAs will hinder the achievement of a system's quality requirements. Since architecture patterns are documented solutions with known properties [2], knowledge of architecture patterns used in a software component can provide valuable insights to component integrators about which QAs are supported, and which are hindered. Previously, the use of architecture patterns has been shown to be an effective and lightweight complementary approach to traditional architecture review methods (e.g., ATAM [3]) to perform architecture reviews [4].

Unfortunately, this information about architecture patterns used in a component is often not readily available, in particular for Open Source Software (OSS) products. OSS products are increasingly used in industry [5], but the quality of products varies widely. Therefore, it is important that OSS *integrators* [6] thoroughly evaluate an OSS product before it is integrated into a system [7]. Documentation of OSS products often

I. Crnkovic, V. Gruhn, and M. Book (Eds.): ECSA 2011, LNCS 6903, pp. 147–163, 2011.

lacks information about their design (including patterns that are used) [8]. The literature provides little guidance to practitioners who would wish to identify architecture patterns. Existing tools for pattern identification are limited to (object-oriented) design patterns, such as documented in [9]. The varying granularity of components (e.g., a class/object as a component versus an executable as a component) makes automated identification of architecture patterns inherently difficult. Reverse engineering methods and tools may help to reverse-engineer the architecture, but do not typically focus on identifying *architecture* patterns [10].

This lack of guidance on how and where to find architecture patterns in OSS products motivated us to investigate how this task can be supported. In our previous work [11], we proposed a conceptual process to streamline the task of identifying architecture patterns in OSS. This paper reports on two additional empirical studies that contribute (a) a validation of the process steps and enhancement of our initial process, and (b) an evaluation of the resulting process. The enhanced process was named IDAPO (IDentifying Architecture Patterns in OSS, pronounced as "Idaho").

This paper proceeds as follows. We present background and motivation in Section 2. Section 3 presents the design history as well as validation and enhancement of IDAPO. Section 4 presents the design and results of a quasi-experiment to evaluate the usefulness of IDAPO. We discuss the findings of the experiment in Section 5. Section 6 concludes and provides an outlook to future work.

2 Background and Motivation

Software Architecture, Patterns and Quality Attributes. Software architecture has been shown to be an important artifact in the software development process [1]. It constitutes a set of architectural design decisions, such as the use of an architecture style or pattern. Most software architectures apply one or more architecture patterns [12]. For instance, architects speak of a 'layered system', or a 'model-view-controller' architecture. Architecture patterns have a documented effect (positive or negative) on a system's quality attributes (QAs) (e.g., performance, reliability) [1, 2, 13]. For instance, a system with layers is likely to be modifiable, as it facilitates a clear separation of concerns. However, passing large numbers of messages up and down the layer 'stack' may cause performance issues [13]. Therefore, one effective way to select OSS products that support the achievement of the system's QAs is to acquire sufficient information of architecture patterns used in those products. Once this information is available, OSS integrators can use the rich information in the pattern documentation (in pattern languages such as [2]) about the potential impact of the pattern's solution on the system QAs [4].

Use and Evaluation of OSS. Over the last decade, an increasing number of software developing organizations is integrating OSS products in component-based development [5, 7]. However, selecting suitable OSS products is a key challenge [8]. To address this, researchers and industry have proposed a variety of OSS evaluation and selection approaches [14]. Typically, these approaches prescribe a list of criteria, such as the level of activity of the OSS community and the number of open bugs, categorized in some categories (e.g., *product*, *community*), on which an OSS product is evaluated. The output is a weighted average of the scores for the criteria. The goal of these

approaches is to provide practitioners with some guidance on the process of evaluating an OSS product. However, these approaches typically do not consider the architectural aspects of a product to assess its impact on a system's QAs. While the abovementioned existing approaches may provide valuable information such as the potential support provided by an OSS product's community, we argue that those methods could be used in tandem with appropriate approaches to identify and understand the architecture patterns used, such as proposed by us. The former assess the maturity of an OSS product, while the latter helps understand the impact on system QAs. While knowledge of architecture patterns is equally important for closed (proprietary) software, we focus our efforts on OSS, since it has become a viable alternative to commercial off-the-shelf (COTS) components [15]. Furthermore, due to the closed nature of COTS components, identifying architecture patterns is virtually impossible for such products.

Pattern Identification. A number of pattern identification techniques and tools have been proposed [9]. However, these techniques and tools focus on the identification of design patterns, which may not have a direct impact on the fundamental structure of a software system [2]. Furthermore, design patterns such as those presented by the Gang of Four [16], are object-oriented, which assumes that the software is written in an object-oriented programming language. These techniques and tools, however, do not support identifying architecture patterns. There are no techniques to automatically identify architecture patterns from source code, and there are a number of obstacles that prevent this. Firstly, there are no commonly accepted formalisms for describing components and connectors between them. Proposed formalisms such as Architecture Description Languages (ADLs) and the UML have several issues [17]: different ADLs focus on modeling different types of systems, and they vary greatly in their expressiveness of software architecture concepts. These obstacles hinder the reverse engineering of source code into a formalism that expresses patterns. The use of reverse engineering tools that could support pattern identification is associated with various challenges [11]. A second obstacle that hinders automated pattern identification is that patterns may be implemented in 'a thousand different ways' [2], and have to be implemented (and customized) according to the specific needs at hand. Therefore, we argue that identifying architecture patterns depends to a large extent on manual techniques. Our proposed process is designed to support this task. In the remainder of this paper, we use the word 'pattern' to refer to *architecture* pattern rather than *design* pattern.

3 Design of the Process

3.1 Design History of IDAPO

We are investigating how practitioners can be guided in the task of identifying architecture patterns. Fig. 1 shows the three empirical studies we have conducted so far.

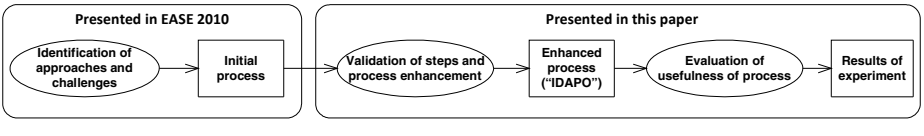

Fig. 1. Overview of research activities and research output. Ovals represent research activities; rectangles represent research outputs.

Previously, we have reported an initial study in which we identified approaches and challenges of 23 master's students that had performed a pattern-identification task in the context of master's courses on Software Architecture and Software Patterns [11]. Based on these findings, we suggested a systematic approach to identify patterns, and presented an initial process definition to support practitioners in this task. As shown in Fig. 1, the current paper reports two additional empirical studies that we conducted. The first (presented in Section 3.2) aimed at validating the process steps and enhancing the process; the second (presented in Section 4) aimed at empirically evaluating the usefulness of our enhanced process (IDAPO) through a quasi-experiment.

3.2 Process Steps Validation and Process Enhancement

In [11] we presented an initial version of a process to support the task of identifying architecture patterns in OSS. In order to validate the different steps of the initial process and enhance the process, we invited all 12 enrolled students who had performed a pattern identification assignment in the context of a master's course on Software Patterns (at the University of Groningen) for a semi-structured interview. Ten students chose to participate. We did not show the participants our initial process in order to prevent getting only confirmatory answers. Instead, we asked the students about the steps taken, their usefulness, what information they had been looking for, obstacles they had encountered, and their "lessons learned", i.e., what steps they would and would not take again.

We digitally recorded the interviews with the participants' consent. Dutch students were interviewed in their native language. The other students were interviewed in English. The interviews lasted 60 minutes on average. All interviews were transcribed verbatim by the interviewer. The Dutch transcriptions were translated into English to allow the other researchers to participate in the data analysis. The data were analyzed using qualitative data analysis techniques [18]. We systematically extracted information about the steps that the students had taken and recorded them in a spreadsheet. We compared the steps to the activities of our initial process presented in [11]. We focused primarily on the steps that students had considered to be useful; for instance, many students considered the use of reverse engineering tools to be a waste of time.

We found that the steps taken by the students mostly corroborated the activities of our initial process. We also found reason to make a number of changes based on new insights gathered from the 10 interviews. While Fig. 2 presents the enhanced process in more detail, in this section we briefly summarize the changes made. We realized that an incremental accumulation of information, such as type and domain of the product as well as implementation technologies used, could be a useful way to identify potentially used patterns, which we refer to as *candidate patterns.* Therefore, we swapped the order of steps (4, 5) with steps (2, 3) in the initial version presented in [11]. A second change we made was to enrich the process with a data flow between the steps, which describes the different pieces of information that can be gathered as a user follows the steps as well as which steps use this information. Thirdly, we used the Business Process Modeling Language (BPMN) to define the process to replace the UML activity diagram notation we used before. This allowed us to more clearly express the different steps of the process. Section 3.3 presents the enhanced process that we named IDAPO.

3.3 IDAPO: A Process for Identifying Architecture Patterns in OSS

A key feature of IDAPO is the idea of incremental accumulation of information. In each step, information regarding the use of patterns in a product is acquired. By systematically recording information about the product's characteristics, a practitioner is encouraged to make details of the product under investigation explicit, which helps the practitioner's analytical thought process. To formalize this idea in IDAPO, we added a 'data flow' to the process, to suggest what information is generated and needed for each step. The resulting definition of IDAPO is shown in Fig. 2 in BPMN. In BPMN, rounded rectangles represent activities; normal lines represent *control flow* (sequence of steps), whereas dotted lines represented *data flow* (which indicate input and output to the various activities). The OSS community is represented by a separate *pool*. In BPMN, a pool represents an organization and is used to border process participants (the default pool is implicit). For more details of BPMN, see [19]. The remainder of this section describes the process steps in more detail; step numbers are enclosed in parentheses.

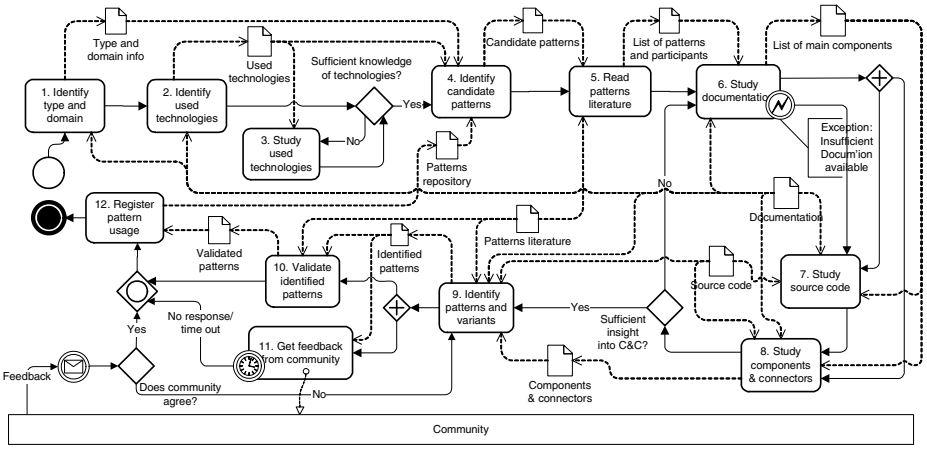

Fig. 2. IDAPO: a process for identifying architecture patterns in OSS

The first step is to (1) **identify the type of software** and its domain. Knowledge of the type and domain of the software may provide hints about the use of certain patterns. For instance, an instant messenger product is likely to use the client-server pattern. Step two is to (2) **identify technologies used** for implementation. If, for instance, CORBA (Common Object Request Broker Architecture) was used, it may be useful to look for the broker pattern. If the process user has insufficient knowledge of technologies, it is advisable to (3) **study those technologies**, which may help in understanding how the system under scrutiny was implemented. Based on the information gathered in previous steps, (4) **candidate patterns may be identified** (i.e., potentially present patterns) and listed. After identifying candidate patterns, (5) the **patterns literature** (e.g. [2]) can be studied to learn more details about those patterns, which will help in recognizing and asserting that the patterns are, in fact, present. The next step is to (6) **study project documentation**, from which insights into the system's architecture,

components and connectors may be gathered (note that the documentation could also be consulted in previous steps). After identifying candidate patterns and studying project documentation, the next step is to (7) **study the source code** and crosscheck with the findings of the documentation. It is important to gain insight into the various (8) **components and connectors** in the system under investigation, since this will help to identify which patterns have been used in the system. Once sufficient information is gathered through studying documentation, source code and components and connectors, the actual (9) **pattern matching** and identification activity starts. This involves comparing the structure and behavior of the pattern to the product's structure. After identification, it is important to (10) **validate the identified patterns** to make sure they have been correctly identified. One way to do this is through peer-review by others (e.g., colleagues). Findings may also be presented to the community for feedback. While the (11) **community may be contacted** earlier to ask for information, our experience has shown that providing some input is more likely to result in a reply. Once identified patterns have been confirmed, the (12) **patterns should be registered** in a patterns repository for later use by others. A few researchers have proposed such repositories for patterns [20] or architectures in general [21]. Over time, the patterns repository will be populated with information of many systems, which we envisage to be a valuable tool for others in understanding the architecture of OSS products.

4 Evaluation of the Process: A Quasi-Experiment

Following the call by Falessi et al. [22] to perform empirical evaluation of new techniques to improve the state of practice in software architecture, we decided to empirically evaluate the usefulness of IDAPO by means of an experiment [23]. We measure usefulness in terms of the number of patterns that are identified. This section is structured following the reporting guidelines for experiments in [24].

4.1 Experiment Goals and Hypotheses

We defined three goals for this experiment. Firstly, we are interested in whether using IDAPO helps to identify more patterns. We argue that the task of *correctly* identifying architecture patterns depends on practitioners' expertise and experience; if IDAPO results in more identified patterns, this expertise is important to assess their correctness. However, not all practitioners have extensive expertise to draw from. In order to be able to more precisely evaluate the usefulness of IDAPO, our second goal was to measure the output in terms of two standard measures: *precision* and *recall* [25]. Thirdly, we wanted to investigate to what extent IDAPO supports the identification of *candidate patterns* based on information gathered in the first three steps. To investigate these goals, we defined six hypotheses, which we discuss next.

To address the first goal, we decided to simply count the number of identified patterns, disregarding whether the patterns are correct or not. IDAPO describes the steps to take, and the information required to identify patterns. Hence the first hypothesis:

H_{01}: Using IDAPO does not change the number of identified patterns.

For all hypotheses, we imply a comparison to the number of patterns identified when not using IDAPO.

Besides looking at the number of identified patterns tested in H_{01}, it is also useful to use standard measures based on a confusion matrix, namely *precision* and *recall* [25]. Precision is defined as the fraction of patterns correctly identified of the total number of identified patterns, i.e., true positives ÷ (true positives + false positives). Recall is defined as the fraction of correctly identified patterns of the total number of correct patterns present, i.e., true positives ÷ (true positives + false negatives). Hence, we defined hypotheses H_{02} and H_{03}.

H_{02}: Using IDAPO does not change the *precision* of *identified* patterns

H_{03}: Using IDAPO does not change the *recall* of *identified* patterns.

As mentioned in Section 3.2, the process emphasizes a step-wise, incremental approach to gather information in a systematic way. In particular, we are interested in the *candidate* patterns based on the first few steps of the process. In order to test this idea, we defined hypothesis H_{04}:

H_{04}: Using IDAPO does not change the number of *candidate* patterns.

Likewise, we decided to also test precision and recall rates for the *candidate* patterns; Hence, we defined to H_{05} and H_{06}:

H_{05}: Using IDAPO does not change the *precision* of *candidate* patterns.

H_{06}: Using IDAPO does not change the *recall* of *candidate* patterns.

For each hypothesis, we imply an alternative hypothesis H_{an} (n=1 to 6) that states that the use of IDAPO *does* result in a higher number of (candidate) patterns.

4.2 Participants and Training

We invited 24 master's students who were enrolled in a course on Software Patterns at the University of Groningen, to participate in our experiment. Participation was not compulsory, but students were advised to participate, as one of the upcoming course assignments would also be to identify patterns in an OSS product in order to perform a pattern-based architecture review [4]; our embedded study with the students was therefore integrated with the course [26]. Fourteen students chose to participate. Table 1 presents demographic information of the participants.

Table 1. Participants of the experiment

Group	ID	Age	Work experience	Degrees	Nationality
Control	P1	24	3½ years, developer	B. (CS)	Netherlands
	P2	27	—	B. (AIM)	Greece
	P3	28	¼ year, developer	B. (CS)	Argentina
	P4	28	—	B. (BI, CS)	Netherlands
	P5	25	1 year, web developer	B. (CS)	Greece
	P6	29	5 years, developer	B. (CS); M. (Psy)	Belgium
	P7	25	—	B. (BI)	South Africa
Treat-	P8	27	2 years, developer	B. (CS)	Netherlands
ment	P9	25	1 year, web developer	B. (CS)	South Africa
	P10	23	2 years, web developer	B. (CS)	Netherlands
	P11	24	5 years, OSS developer	B. (CS)	Netherlands
	P12	25	2 years, web developer	B. (CS)	Spain
	P13	22	—	—	Netherlands
	P14	21	—	B. (CS)	Netherlands

Section 4.4 discusses the assignment procedure to the groups. The average age of the control group was almost 24, whereas the average age of the treatment group was approximately 26½. Note that work experience should be interpreted as part-time jobs. One participant (P11) actively contributed to a small OSS project. All but one participant (P13) had finished a bachelor's (B) degree in computer science (CS), bio-informatics (BI) or applied informatics and multimedia (AIM). One participant (P6) also had a master's (M) degree in psychology (Psy). Most of them had varying levels of expertise in different topics, as listed in Table 2, e.g., three participants assessed themselves as having *advanced* knowledge of software engineering.

When we conducted the experiment, the students had attended six 2-hour lectures of the 8-week course on Software Patterns. All students also had followed a course on Software Architecture. The data presented in Tables 1 and 2 were gathered through a pre-study questionnaire the day before the experiment.

Table 2. Participants' self-assessed levels of expertise

Topic	None	Beginner	Intermediate	Advanced	Expert
Software engineering	0	5	6	3	0
Software architecture	0	6	7	1	0
"Gang of Four" design patterns	3	5	5	1	0
Architectural patterns	0	10	4	0	0
Development process of OSS	5	5	4	0	0
Experience w. integrating COTS	4	3	4	2	1

4.3 Task and Materials

The task given to the participants was to identify as many architectural patterns in a specified OSS project as possible: the JBoss application server. We selected JBoss for three reasons. Firstly, it is an industry-strength system (no 'toy' project), which is widely used in industry. Secondly, we expected that the participants would be able to find sufficient information about this product in the limited available time, since it is well known and extensive documentation is available. Thirdly, we already had insight into the architectural patterns used in this product, which we would need as a marking scheme for assessing the number of *correctly* identified patterns as well as the precision and recall. Participants in both groups were handed out the assignment form. The treatment group was given two additional instruments: (1) our process as shown in Fig. 1 accompanied by a description of each step; and (2) a simple spreadsheet template to record information found in each step. Additionally, the participants had access to the five volumes of the Pattern-Oriented Software Architecture (POSA) series of books (e.g., [2]), which list various software patterns.

4.4 Experiment Design

The experiment design was a between-subjects design, to compare results from a control group and a treatment group. Based on our previous experience in conducting research with students, we expected that the participants would have varying levels of experience and expertise. Since this would have constituted a threat to the outcome of the experiment, we decided to non-randomly assign participants to the control and treatment groups. Hence, this experiment was a *quasi*-experiment [27, 28]. Eight

participants had indicated to have other course obligations in either the morning or afternoon of the day of the experiment; based on this information, three participants were assigned to the control group, and five were assigned to the treatment group. Based on the information about work experience gathered in the pre-study questionnaire, we assigned the remaining six students, resulting in two equally sized and approximately equivalent groups (see Table 1).

The *treatment*, or *independent variable* manipulated by this study is the reference process, with one treatment: IDAPO is provided, and one control: IDAPO is not provided. The *dependent variable* is the number of architecture patterns identified by the participants using and not using the process.

4.5 Experiment Procedure

We conducted the experiment in two sessions. The control group performed the task in the morning session, and the treatment group (provided with IDAPO) was invited for the afternoon session. This order ensured that the control group did not see IDAPO (to prevent the diffusion or imitation of treatment threat [29]). In both sessions, the researcher gave a brief introduction (15 min) to explain the background and motivation of identifying patterns. For the treatment group, the researcher also explained the different steps of IDAPO. Both groups were given two hours for this task. One participant in the control group had to leave 30 minutes early due to other course obligations (P1). After the two hours, participants were asked to fill out a post-study questionnaire; we used separate post-questionnaires for the two groups, as only the treatment group could be asked about their experiences with IDAPO.

4.6 Analysis and Results

4.6.1 Establishing a Set of Trusted Patterns

In order to be able to determine precision and recall measures, we need to compare the findings to a certain set of "correct" patterns of which we are confident that they are present in the product. In order to establish such a trusted subset of patterns, we used three different sources. Firstly, we used a research report that presents an analysis of the JBoss architecture (v.2.2.4, 2002) [30]. Secondly, we used a technical report (from 2005) that reports on the architecture recovery of JBoss [31]. Thirdly, we used a report from a previous group that had identified patterns in JBoss in the context of the 2009 edition of the Software Patterns course (mentioned in [11]); one of its authors had extensive professional experience as an administrator of JBoss. Table 3 lists patterns identified by the different sources, as well the patterns that we decided to include in the trusted subset.

We made this selection based on the reports, which described the patterns and their location in JBoss, as well as our level of confidence that we had in the presence of these patterns. During our selection, we also considered that the different sources have studied different versions of JBoss. We could not find sufficient justification to include the *Pipes-Filters* and the *Factory* patterns. The column 'Trusted' indicates which patterns are included in the trusted subset. We listed all patterns identified (for both control and treatment group) in a spreadsheet. In order to calculate precision and control measures, we counted only those patterns that were listed in the trusted list (Table 3).

Table 3. Derivation of a trusted subset of patterns

Pattern	Liu	Salehie et al.	Report 2009	Trusted
Microkernel	Yes	Yes	Yes	Yes
Layers	Yes	Yes	-	Yes
Pipes & Filters	Yes	-	-	-
Broker	Yes	-	Yes	Yes
Dynamic proxy	Yes	-	Yes	Yes
Proxy	Yes	Yes	-	Yes
Interceptor	Yes	Yes	Yes	Yes
Client/server	-	-	Yes	Yes
Active repository	-	-	Yes	Yes
Factory	-	-	Yes	-

4.6.2 Descriptive Statistics

Table 4 presents the descriptive statistics of the results. We counted the number of identified patterns of the control and treatment group as a whole. The first three columns list the results when counting *all* patterns, disregarding their correctness; column 1 lists the total number of patterns of the control group (18); column 2 lists the total number of candidate patterns identified by the treatment group ('T. candid.', 36), and column 3 lists the total number of identified patterns (as output of step 9 in the process, see Fig. 2) listed by the treatment group ('T. final', 16).

Table 4. Number of patterns per group, mean and standard deviation. Columns 1-3 consider all patterns identified; columns 4-6 only consider the trusted patterns.

	Counting all patterns			Counting trusted patterns only		
	(1) Contr.	(2) T. candid.	(3) T. final	(4) Contr.	(5) T. candid.	(6) T. final
Total	18	36	16	10	21	10
Mean	2.6	5.1	2.3	1.4	3.0	1.4
Std. dev	2.1	2.3	2.4	0.9	1.1	1.8

Columns 4-6 show the results similarly as column 1-3, but only taking the *trusted* patterns into account (resulting in 10, 21 and 10 patterns, respectively). When counting trusted patterns only, there is no difference between the control group and the final results of the treatment group. The treatment group as a whole identified 21 candidate patterns, which suggests the treatment group was on the right track. Fig. 3 shows boxplot diagrams for the results presented in columns 1-6.

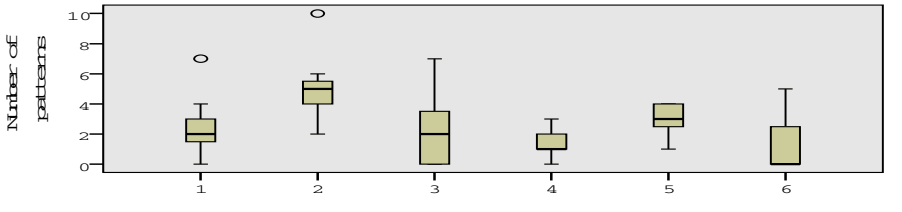

Fig. 3. Distribution of numbers of identified patterns by the control group, treatment group (candidate and final). Boxplots 1-3: counting all patterns, corresponding to columns 1-3 in Table 9. Boxplots 4-6: trusted patterns only (corresponding to columns 4-6 in Table 4).

Table 5 presents the mean and standard deviation values of the *precision* and *recall* rates, calculated for the control group and the treatment group. For the latter, we calculated precision and recall both for the candidate results and the final results. Table 5 shows that the average precision of the control group is 0.56 with an average recall of only 0.18. This suggests that about half of the control group's patterns are correct, but that (on average) less than 20% of the trusted patterns were identified. The *candidate* results of the treatment group score better, with a precision of 0.62 at a recall rate of 0.37, suggesting that (on average) the treatment group identified more correct candidate patterns. However, when looking at the *final* results of the treatment group precision is only 0.30 at a recall of 0.18, worse than the control group. The relative high values for the standard deviations of precision (0.35) and recall (0.23) for the treatment group's final results suggest a large variation among participants. We found that three participants in the treatment group did not list any "final" patterns (as opposed to one participant in the control group).

Table 5. Precision and recall for control, treatment candidate and treatment final results

	Control		Treatment			
			Candidate patterns		Final	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Mean	0.56	0.18	0.62	0.38	0.30	0.18
Std. dev.	0.32	0.11	0.21	0.13	0.35	0.23

4.6.3 Results of Statistical Analysis

We performed statistical analyses on the number of identified patterns and the calculated precision and recall rates to test the six hypotheses. The assumptions underlying parametric tests such as the t-test were not fulfilled [32], since the data contained outliers and we could not assume that the data have a normal distribution. Therefore, we decided to use the Mann-Whitney U test, which is a non-parametric alternative to the t-test for two independent samples [32]. Table 6 lists the p-values for each of the six hypotheses. The columns 'Candidate' and 'Trusted' indicate whether the hypotheses consider the candidate patterns (of the treatment group) and whether only trusted patterns were counted, respectively. We reject a hypothesis if the p-value is less than the significance level of α=0.05. We used SPSS version 18 for all statistical tests.

Table 6. Hypotheses and resulting p-values of the Mann-Whitney U test

Hyp.	Variable	Candidate	Trusted	P-value	Decision
H_{01}	Number of identified patterns	No	No	0.435	Retain H_{01}
H_{02}	Precision of identified patterns	No	Yes	0.324	Retain H_{02}
H_{03}	Recall of identified patterns	No	Yes	0.597	Retain H_{03}
H_{04}	Number of candidate patterns	Yes	No	**0.051**	Retain H_{04}
H_{05}	Precision of candidate patterns	Yes	Yes	0.555	Retain H_{05}
H_{06}	Recall of candidate patterns.	Yes	Yes	**0.026**	**Reject H_{06}**

Table 6 shows that we could not find compelling evidence to reject hypotheses H_{01}-H_{05} (all p-values > α=0.05). In other words, there was not sufficient evidence to con-

clude that using IDAPO resulted in a higher number of identified patterns (H_{01}), a higher precision of identified patterns (H_{02}), a higher recall of identified patterns (H_{03}), a higher number of candidate patterns (H_{04}), and a higher precision of candidate patterns (H_{05}). With respect to H_{04}, we found that there is some evidence ($p=0.051$) that using IDAPO results in a higher number of *candidate* patterns, but since the p-value is smaller than our significance level ($\alpha=0.05$) we do not reject H_{04}. On the other hand, we found evidence ($p=0.026 < 0.05$) to **reject** hypothesis H_{06} ('using IDAPO does not change the recall of candidate patterns'). Together, these results suggest that IDAPO helps to improve the **recall** of candidate patterns.

4.6.4 Results of Post-study Questionnaires

The post-study questionnaire questions were rated using a five-point Likert scale, ranging from Totally Disagree (TD), to Disagree (D), Neutral (N), Agree (A) and Totally Agree (TA).

Results of Treatment Group. Table 7 presents the results for the treatment group. Numbers indicate the number of participants that gave a certain rating, e.g., two subjects answered 'Neutral' on question T1.

Table 7. Post-study questionnaire results for the treatment group. High scores are highlighted

ID	Question	TD	D	N	A	TA
T1	I followed the process step by step in the order prescribed.	1	1	2	1	2
T2	Identifying the type and domain of the software is helpful.	0	1	2	2	2
T3	Identifying the used technologies is helpful to identify patterns.	0	1	1	1	4
T4	The process helped me to identify patterns that I wouldn't have found otherwise.	0	4	2	1	0
T5	The suggested order of steps in the process made sense.	0	2	2	1	2
T6	Storing information per step in the spreadsheet was useful.	0	2	2	1	2

Table 7 shows that the degree to which the subjects followed IDAPO varied (T1). This suggests that participants disliked the process rigid order of steps, and would like to have more flexibility. The results for T2 suggest that most subjects agree that identifying the type and domain of the software is helpful. The results show that identifying used technologies (T3) was considered to be very helpful. Most participants did not think that IDAPO helped to identify patterns that could not have been identified otherwise (T4). Two participants were undecided, and only one participant agreed. Participants were divided on whether the order of IDAPO's steps was sensible (T5). Also, participants were equally divided on whether storing information per step in a spreadsheet was useful (T6).

We also gathered results from a few open questions. Some suggestions were:

- A spreadsheet was not considered suitable to record intermediate information;
- The process should be made less sequential;
- After identifying type and domain, always read documentation to learn about components and used technologies.

The main challenges encountered by the treatment group were:

- To find the right documentation and information;
- Lack of time to read source code, and also to identify patterns;
- Unfamiliarity with JBoss.

Results of Control Group. Table 8 lists the questions and the scores for the control group. The results for C1 show that most participants either disagreed or were undecided on whether they knew what steps to take to identify architecture patterns. Only one participant indicated he knew what approach to take. This confirms our assertion that there is a need to provide some guidance in this task. The second question (C2) was to find out whether participants found sufficient information to identify patterns. Again, most participants indicated disagreement or neutrality. This suggests that, in general, there is a need to identify useful sources of information.

Table 8. Post-study questionnaire results for the control group

ID	Question	TD	D	N	A	TA
C1	I knew what steps to take to perform the assignment.	0	2	4	1	0
C2	I found sufficient information to identify patterns.	1	2	2	2	0

We also asked the participants for suggestions for improvement as well as challenges encountered. Some suggestions included:

- Use of a debugger to trace the execution to find relations among components;
- Search for images of the architecture that may lead to useful sources.

The main challenges encountered by the control group were:

- Unfamiliarity with JBoss; getting to know the system;
- Studying the source code is like finding a needle in a haystack;
- Finding the right information; inconsistent documentation; pattern naming is inconsistent (e.g. a 'proxy' component implementing the 'dispatcher' pattern)

5 Discussion

In this section we interpret the findings from the experiment, the post-study questionnaires and discuss implications for further research. The overall motivation for the development of IDAPO was to provide guidance to practitioners in identifying architecture patterns in an OSS product. While we did not find evidence that the use of IDAPO helped to identify more architecture patterns than the control group who did not use IDAPO, the results showed some modest advantages. The results of the experiment suggest that the use of IDAPO helped to identify more candidate patterns that were considered correct (based on a set of "trusted patterns" derived in subsection 4.6.1). This means that many potentially present patterns identified based on information about the product's type, domain and used technologies (steps 1 and 2 in Fig. 2) turned out to be correct. Questions T2 and T3 in Table 7 confirm that participants considered these to be useful steps. These results suggest that IDAPO is helpful to identify candidate patterns. The use of IDAPO also helped to improve the recall of the candidate patterns, which indicates that compared to the control group, a larger number (on average) of correct patterns were recovered.

On the other hand, when considering only the *final* results of the treatment group, we did not find any evidence that the use of IDAPO resulted in more identified patterns than the control group (H_{01}), nor in a higher precision (H_{02}) or recall (H_{03}). This suggests that, while IDAPO helps to identify candidate patterns, the process does not provide sufficient guidance to assess that the patterns are in fact present. Questions T1, T4 and T5 in Table 7 seem to confirm this; participants did not follow the steps in the suggested order (T1), participants did not think that IDAPO exclusively helped to identify certain patterns (T4), and participants were divided on whether the order of steps made sense (T5). Based on these observations as well as the results of the post-study questionnaires of both the control and treatment groups, we point out the strong points of IDAPO as well as points for improvement. IDAPO is useful for guidance, but should not be prescriptive. Rather, a *process framework* seems to be more appropriate, from which a user can select appropriate activities to derive a process that is tailored to the context and needs of the user. This also allows prioritizing tasks in case that time is limited. Furthermore, it is important to investigate ways that a user can get familiar with a system more quickly as well as approaches to find appropriate documentation and information. Better ways to record intermediate information that support the user to manage this information and draw appropriate conclusions should be investigated. The use of tools (e.g., debuggers) could provide additional ways to acquire more information of a system's structure. Through our interviews we found that the use of tools was often not very helpful; therefore, we emphasize that it is important to understand how tools can provide support and what type of information can be acquired.

5.1 Threats to Validity

Conclusion Validity. The number of subjects is a threat to conclusion validity. Fourteen subjects were willing to participate in our experiment, which were divided into two groups (control, treatment) of seven. However, we did not intend to make conclusive statements based on this single experiment only. Rather, our results should be considered exploratory and help us to gain insights in the usefulness of IDAPO. **Construct validity**. There are a few limitations to construct validity. Firstly, we limited the total time for identifying patterns to two hours. This limitation has a direct effect on the amount of work that can be done, and therefore on the number of patterns that can be identified. Participants of the treatment group may have had to spend relatively much time on understanding the steps of IDAPO. However, we chose to limit the time duration in order to be able to recruit a sufficient number of subjects; as the time duration of an experiment increases, fewer participants will be willing to participate. **Internal validity**. We discuss a number of threats to internal validity, which is concerned with the degree to which a change of the dependent variable can be ascribed to a change of the independent variable. The first is instrumentation: the process description and diagram of IDAPO may not have been easy to understand by the treatment group. Though we explained the process to the treatment group, participants may not have fully understood the steps to take. Another instrumentation threat is our marking scheme for assessing "correct" patterns. This instrument (discussed in subsection 4.6.1) was used to perform calculations of precision and recall. Our conclusions depend on the extent to which we correctly confirmed the patterns. We derived this trusted subset of patterns based on three independent sources. However, the three re-

ports do not fully agree on the patterns. In order to decide which patterns to include in the trusted subset, we have (a) studied the description of how the patterns were implemented, thereby assessing the credibility of the description and the pattern's usage, and (b) attempted to find additional information through web searches in order to be able to confirm them. It is noteworthy that the patterns listed by the three sources that we could not confirm (and therefore were not included in the trusted subset) were not identified by either group. Besides this, JBoss may contain patterns that have not been listed by any of the three reports, which means that these are not included in our trusted subset. The second threat is that of selection: the control and treatment groups may not be as equivalent as we intended in terms of work experience and knowledge of related topics (see Table 1). Furthermore, the average age in the control group was 2.5 years higher (26½) than the average age in the treatment group (almost 24); this difference suggests that the control group has a few more years of experience in the field of software engineering. This could have negatively biased the results of the treatment group, which strengthens the decision to reject hypothesis H_{06}. **External validity**. Threats to external validity are those that may limit the applicability of the results to industry practices. The use of master's students as subjects is an important factor that deserves attention, and has been discussed in the literature [26, 33, 34]. We do not consider the use of students to be a major threat, since it is not yet a common practice in industry to identify patterns in an OSS product. Some researchers mention that students are suitable to be used to evaluate new techniques [35]. Furthermore, since the participants were master's students (rather than undergraduates), they can be considered to be 'novice' professionals. A potential threat to external validity is that a treatment is applied on a 'toy' problem. However, we selected the JBoss application server for this experiment, which is an industrial-strength software product.

6 Conclusion and Future Work

In this paper we present and evaluate IDAPO: a process that provides guidance to practitioners who wish to identify architecture patterns in an OSS product. The process design is based on empirically identified steps. We have conducted a quasi-experiment to empirically evaluate IDAPO. We found evidence that the first few steps of IDAPO are particularly helpful to identify candidate patterns (potentially present in the product). We believe that IDAPO can be a valuable contribution to the toolkit of practitioners who need to evaluate OSS products. However, the results also suggested that the other steps of IDAPO could be improved. We believe that the process steps should become more flexible, and become part of a *process framework*, which can be tailored to the user's needs. Furthermore, it would be valuable to investigate how IDAPO can support identification of architectural *tactics*, such as documented in [1]. Tactics support the achievement of quality attributes and can therefore provide valuable insights similar to the information conveyed by patterns.

Acknowledgments. This work is partially funded by IRCSET under grant no. RS/2008/134 and by Science Foundation Ireland grant 03/CE2/I303_1 to Lero.

References

[1] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley, Reading (2003)

[2] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-oriented Software Architecture - A System of Patterns. J. Wiley and Sons Ltd., Chichester (1996)

[3] Kazman, R., Klein, M., Barbacci, M., Longstaff, T.: The Architecture Tradeoff Analysis Method. In: ICECCS, pp. 68–78 (1998)

[4] Harrison, N.B., Avgeriou, P.: Pattern-Based Architecture Reviews. IEEE Software (2011) (in Press)

[5] Hauge, Ø., Ayala, C., Conradi, R.: Adoption of Open Source Software in Software-Intensive Organizations - A Systematic Literature Review. Inf. Softw. Technol. 52(11), 1133–1154 (2010)

[6] Hauge, Ø., Sørensen, C.-F., Røsdal, A.: Surveying Industrial Roles in Open Source Software Development. In: Int'l Conf. on Open Source Systems, pp. 259–264 (2007)

[7] Ruffin, C., Ebert, C.: Using open source software in product development: A primer. IEEE Software 21(1), 82–86 (2004)

[8] Stol, K., Ali Babar, M.: Challenges in Using Open Source Software in Product Development: A Review of the Literature. In: 3rd FLOSS Workshop, ICSE 2010, pp. 17–22 (2010)

[9] Dong, J., Zhao, Y., Peng, T.: Architecture and Design Pattern Discovery Techniques - A Review. In: Int. Conf. Softw. Eng. Research and Practice, pp. 621–627 (2007)

[10] Tonella, P., Torchiano, M., du Bois, B., Tarja, S.: Empirical studies in reverse engineering: state of the art and future trends. Empir. Software Eng. 12(5) (2007)

[11] Stol, K., Avgeriou, P., Ali Babar, M.: Identifying Architectural Patterns Used in Open Source Software: Approaches and Challenges. In: EASE, Keele, UK (2010)

[12] Shaw, M., Garlan, D.: Software Architecture: Perspectives on an Emerging Discpline. Prentice-Hall Inc., Englewood Cliffs (1996)

[13] Harrison, N.B., Avgeriou, P.: Leveraging Architecture Patterns to Satisfy Quality Attributes. In: Oquendo, F. (ed.) ECSA 2007. LNCS, vol. 4758, pp. 263–270. Springer, Heidelberg (2007)

[14] Stol, K., Ali Babar, M.: A Comparison Framework for Open Source Software Evaluation Methods. In: Int'l Conf. on Open Source Systems, pp. 389–394 (2010)

[15] Fitzgerald, B.: The transformation of open source software. MISQ 30(3) (2006)

[16] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)

[17] Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languages. Trans. Softw. Eng. 26(1), 70–93 (2000)

[18] Seaman, C.B.: Qualitative methods in empirical studies of software engineering. Trans. Softw. Eng. 25(4), 557–572 (1999)

[19] White, S.A.: Introduction to BPMN, BPTrends (July 2004)

[20] van Heesch, U.: `http://www.cs.rug.nl/search/ArchPatn/OpenPatternRepository`

[21] Booch, G.: `http://www.handbookofsoftwarearchitecture.com` (accessed December 5, 2010)

[22] Falessi, D., Ali Babar, M., Cantone, G., Kruchten, P.: Applying empirical software engineering to software architecture: challenges and lessons learned. Empir. Software Eng. 15(3), 250–276 (2010)

[23] Wohlin, C., Höst, M., Henningsson, K.: Empirical Methods and Studies in Software Engineering. LNCS, pp. 145–165 (2008)
[24] Jedlitschka, A., Pfahl, D.: Reporting Guidelines for Controlled Experiments in Software Engineering. In: ISESE, pp. 95–104 (2005)
[25] Olson, D.L., Delen, D.: Advanced Data Mining Techniques. Springer, Heidelberg (2008)
[26] Carver, J.C., Jaccheri, L., Morasca, S., Shull, F.: A checklist for integrating student empirical studies with research and teaching goals. Empir. Software Eng. 15(1) (2010)
[27] Kampenes, V.B., Dybå, T., Hannay, J.E., Sjøberg, D.I.K.: A Systematic Review of Quasi-Experiments in Software Engineering. Inf. Softw. Technol. 51(1), 71–82 (2007)
[28] Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., el Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. Trans. Softw. Eng. 28(8), 721–734 (2002)
[29] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic, Dordrecht (2000)
[30] Liu, J.: Research Project: An Analysis of JBoss Architecture, `http://www.huihoo.org/jboss/jboss.html` (accessed March 2, 2011)
[31] Salehie, M., Li, S., Tahvildari, L.: 'Architectural Recovery of JBoss Application Server', Tech. Report no. UW-E&CE#2005-02, University of Waterloo (2005), `http://stargroup.uwaterloo.ca/~s7li/publications/ieee_papers/uw-tr-1.pdf`
[32] Hollander, M., Wolfe, D.A.: Nonparametric statistical methods, 2nd edn. John Wiley & Sons, Inc., Chichester (1999)
[33] Höst, M., Regnell, B., Wohlin, C.: Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. Empir. Software Eng. 5(3), 201–214 (2000)
[34] Svahnberg, M., Aurum, A.K., Wohlin, C.: Using Students as Subjects – An Empirical Evaluation. In: ESEM, Kaiserslautern, Germany, pp. 288–290 (2008)
[35] Berander, P.: Using students as subjects in requirements prioritization. In: ISESE (2004)