Assessing the value of Architectural Information Extracted from Patterns for Architecting

Muhammad Ali Babar, Barbara Kitchenham, Piyush Maheshwari
National ICT, Australia, University of New South Wales, Australia
{malibaba, barbara.kitchenham, piyush.maheshwari}@nicta.com.au

Background: We have developed an approach to identifying and capturing architecturally significant information from patterns (ASIP), which can be used to improve architecture design and evaluation.

Goal: Our goal was to evaluate whether the use of the ASIP provides more effective support in understanding or designing software architecture composed of the software design patterns which are the source of the ASIP compared with the original design pattern documentation.

Experimental design: Our subjects were 20 experienced software engineers who had returned to University for a post graduate course. All participants were taking a course in software architecture. The participants were randomly assigned to two groups of equal size. Both groups performed two tasks: understanding the use of J2EE design pattern in a given architecture based on the quality requirements the architecture was supported to satisfy, and designing software architecture to satisfy a given set of quality requirements using J2EE design patterns. For the first task, one group (treatment group) was given ASIP information the other (control group) was given the standard J2EE pattern documentation. For the second task, treatment group became the control group and vice versa and the type of support information was kept constant. The outcome variables were the number of correctly identified design patterns. The participants also completed a post-experiment questionnaire.

Result: The average score for the first task for the treatment group was 23.90 and for the control group was 13.80. The difference between the groups was significant using Mann-Whiney test ($p=0.0375$). The average score for the second task for the treatment group was 26.85 and for the control group was 19.60. Mann-Whitney test revealed that the difference between the groups was again significant at ($p=0.035$). Post-study questionnaire revealed that 18 of the 20 participants believed that the ASIP was more helpful than pattern documentation for understanding and designing architectures.

Conclusion: Our results support the hypotheses that ASIP information is more helpful in understanding or designing software architectures using software design patterns than pattern documentation itself.

Keywords: Controlled experiments, software architecture design and evaluation, design patterns.

1. INTRODUCTION

Software Architectures (SA) of large and complex systems are usually composed of architecture styles and patterns, which can help to achieve one or more quality attributes but may hinder others [1-3]. Scenarios are an effective means of characterizing the quality goals of a system [4]. Patterns are then used to achieve those goals. Thus, scenarios, quality attributes, and patterns are architecturally significant artefacts¹, which form a synergistic relationship that should be explicitly codified to improve architecting activities [5].

Recently, there have been attempts to codify the links that exist among scenarios, quality attributes, and patterns [6-8]. However, these approaches identify and link different pieces of architectural information (scenarios, quality attributes, and patterns) from various sources with the exception of patterns themselves². There has been little research on exploring patterns for architecturally significant artefacts and the relationships among them that exist informally in each pattern's description. Moreover, we have found that the existing formats of pattern documentation are not appropriate for explicating the schemas of the relationships among scenarios, quality attributes, and patterns in a way that makes such information readily reusable. This means that there is little use/reuse of the architecturally significant artefacts (such as general scenarios, quality attributes and tactics) informally described in pattern documentation [9].

In order to distil and explicitly document the relationships of scenario, quality attributes, and patterns in a way that makes such information readily reusable, we have developed a framework for "mining patterns". This framework consists of a process of exploring patterns for architectural information, guidelines to identify several types of architectural information in a pattern's description, and template to document the extracted information. The extracted information documented in the given template is called Architecturally Significant Information extracted from Patterns (ASIP). Architecturally significant information is information that can support decision making during

¹The artefacts used or generated to support architecture design or assessment process.

² Section 2 briefly describes how these approaches link patterns and quality attributes through scenarios.

architecture design and assessment. We have undertaken a research program to evaluate the effectiveness of the pattern-mining process and the value of the resulting information. The program involved an observational study and two experiments which collected objective and subjective data. We reported the results of the observational study in a previous paper [9] and the results of the first experiment have been reported in [10]. In this paper we report the results of the second experiment. This experiment involved two separate experimental tasks: architecture comprehension and architecture design.

The remainder of this paper is organized as follows. Section 2 describes the background and motivation of our research. Section 3 describes the ASIP. Section 4 delineates the potential usefulness of the *ASIP* for software architecting. The experimental design and procedures are explained in Section 5. The experimental results and analysis are presented in Section 6. Section 7 briefly discusses the results and Section 8 concludes the paper.

2. BACKGROUND AND MOTIVATION

Our framework for mining pattern for architecturally significant information is based on the relationship between quality attribute, quality related scenarios, and pattern. A quality attribute is a non-functional requirement of a software system, e.g., security, flexibility, changeability, portability, and so forth. According to [11], software quality is the degree to which the software possesses a desired combination of attributes. Quality attributes of large software intensive systems are usually determined by the system's software architecture. It is widely recognized that SA of large systems put constraints on the achievement of desired set of quality attributes such as variability, flexibility and others [1]. Since SA plays a vital role in achieving quality attributes, it is important to focus on quality related issues during architecture design and to assess the capability of the designed architecture with respect to the desired quality attributes as early as possible.

A scenario is a textual, system independent specification of a quality attribute [1]. Scenarios are considered an effective and useful mechanism for specifying quality attributes. Several approaches use scenarios to encourage disciplined thinking during architecture design and evaluation activities [12, 13]. Scenarios are very flexible. They can be used for systematically reasoning about or evaluating most quality attributes. For example, we can use scenarios that represent failure to reason about the appropriate architectural mechanisms to satisfy availability and reliability requirements; scenarios that represent change requests can be used to examine modifiability; scenarios that represent threats can help select suitable security tactics; or scenarios that represent ease of use can support usability assessment. Moreover, stakeholders need to define scenarios at a detailed (concrete) level so that their impact can be fully understood [14].

A pattern is a known solution to a recurring problem in a particular context. Patterns provide a mechanism of documenting and reusing design knowledge [15]. The software architecture of complex and large system is usually developed using several different patterns. Architectures of such systems evolve by successively integrating different patterns that may be described at different levels of abstraction, to deal with particular design problems [16]. One of the main goals of using patterns is to design an architecture with known quality attributes [2]. Each pattern either supports or inhibits one or more quality attributes.

There are several formats for documenting patterns. However, most pattern documentation formats include at least four sections: context, problem, solution and quality consequences. Each pattern's description contains a large amount of architectural information, e.g., scenarios, quality attributes supported or hindered, tactics, and others. The scenarios informally embedded in a pattern description can be used to characterize the quality attributes achieved by that pattern.

There has been some effort to make these relationships explicit and codify this knowledge [7, 8]. For example, Bass and John associate architectural patterns with usability through scenarios by generating scenarios that characterize usability and presenting architecture patterns that can satisfy those scenarios [7]. Folmer and Bosch [8] also link usability with patterns by decomposing usability attributes into usability properties, which can be characterized by scenarios but they do not provide such scenarios. However, these approaches do not attempt to systematically capture and document the relationships among scenarios, quality attributes, and patterns informally described in pattern documentation. We claim that a disciplined approach to extract and document such relationships from patterns can help software engineers to fully utilize the large amount of architecturally significant information implicitly described in patterns' documentation in order to achieve desired quality attributes.

3. WHAT IS THE ASIP

Our research efforts have concentrated on improving the architecture design and evaluation processes by understanding current techniques and practices, identifying problem areas, developing appropriate complementary or alternative techniques and assessing them by accumulating empirical evidence. Pattern-mining is one of the techniques we have developed to support software architecting process.

The main argument underpinning this line of research is that the existing formats of pattern documentation contain a large amount of information (i.e. usually more than a dozen pages), which may not be necessarily helpful during software architecting. Rather, such a verbose documentation format makes it hard to gain in-depth knowledge of a pattern [17]. Moreover, it may hinder the full utilization of the large amount of architecturally significant information implicitly described in a pattern's documentation.

Architecturally significant information is information that can support decision making during architecture design and evaluation. For example, general scenarios that help stakeholders understand the potential effect of using a certain pattern on certain quality attributes or help architects to select a particular analysis model to assess the

level of support provided by architecture decision for one or more quality attributes, or understand the rationale for using a particular pattern and the tactics that a pattern applies to satisfy various quality attributes. Such information is usually informally described in a pattern's documentation. Moreover, a pattern, quality attributes affected (positively or negatively) by that pattern, and general scenarios for characterizing those quality attributes form a synergistic relationship [5]. This relationship helps stakeholders understand existing pattern-based architectures or identify suitable patterns for satisfying certain quality requirements in a new architecture. However, this synergistic relationship is not explicated in existing pattern description formats. We have demonstrated that architecturally significant information informally described in a pattern can be captured in a format that presents the synergistic relationship between pattern, quality attributes, and scenarios, at a level of abstraction appropriate for software architecting [9]. The extracted information is maintained in the *ASIP* template as shown in Table 1.

TABLE 1 THE ASIP CAPTURED FROM J2EE FRONT CONTROLLER PATTERN

Pattern Name: <i>Front Controller</i>		Pattern Type: <i>Design pattern</i>	
Brief description		<i>This pattern holds the common processing logic within the presentation tier. It handles requests and manages content retrieval, security, and navigation, delegating to a Dispatcher...</i>	
Context		<i>Presentation processing requires complex view navigation, creation and delivery....</i>	
Problem description		<i>A centralized access point is required for presentation request handling to support the integration of system services. Each view provides its own system services, which results in duplicate code....</i>	
Suggested solution		<i>Use a Controller as the initial point of contact for handling a request for services like authentication and authorization, delegating business processing, handling errors...</i>	
Forces		<i>Logic that is best handled in one location is replicated within numerous Views. Decision points exist for retrieving and manipulating data. A user has direct access to each service, which causes complexity. This pattern provides a user with a centralized means of access different services.</i>	
Available tactics		<i>Servlet front, JSP front, Command and controller, Physical/ logical/Multiplex resource mapping....</i>	
Affected Attributes		Positively	Negatively
		<i>Reusability, Manageability of security, centralized services.</i>	<i>Reduced efficiency, Another level of indirection.</i>
General scenarios	S1	<i>Minimize code duplication for system services like data verification, user authentication and others.</i>	
	S2	<i>System services are managed by a centralized control mechanism, which may invoke them as needed.</i>	
	S3	<i>A system shall control and coordinate several requests from the same user in the same session.</i>	
Usage examples		<i>Logging users' actions, keep tracking user's shopping carts</i>	

4. THE VALUE OF THE ASIP

Compared with standard pattern documentation, the *ASIP* template (Table 1) captures and presents different pieces of a pattern's description in a succinct format at an abstraction level suitable for software architecting. At this stage general scenarios are used to characterize quality attributes and suitable patterns are chosen based on their support for the required quality attributes. Linking pattern, quality attribute, and general scenarios forms a coherent representation of architecture design knowledge, which should be represented in a manner that improves the usability of such knowledge. Table 1 demonstrates how the *ASIP* extracted from one of the core J2EE design patterns, Front Controller pattern [18], is represented in a manner that makes the relationships among scenarios, quality attributes, and patterns explicit. In particular, it captures "forces" which is one of the most important parts of a pattern description. The forces of a pattern are usually described implicitly in most of the pattern documentation styles. Recently, there are efforts to pay more attention to the forces of a pattern [6, 19].

The forces of a pattern describe the factors that can cause a problem if they interfere with one another. The pattern attempts to resolve clashes among those factors. Discussion of forces also captures tradeoffs made in a pattern. Moreover, the forces of a pattern also describe the rationale for using that particular pattern, which helps understand the implications of using a particular pattern in software architecture design. Based on our experiences with the *ASIP*, we believe that instead of reading several pages of pattern documentation to comprehend a pattern's rationale, the forces section of the *ASIP* template (Table 1) presents a pattern's rationale in a precise manner that can help gain clear understanding of a pattern's rationale effectively and quickly.

The abstract information captured in the *ASIP* template can be concretized for a specific project. For example, the scenarios extracted from the Front Controller pattern can be concretized to specify modifiability or reusability quality requirements of a project. For instance, the scenario S1 in the *ASIP* template "Minimize code duplication for system services like data verification, user authentication and others." can be concretized for specifying easily modifiable security policies in this way: "Application should have a centralized place to implement organization's security policies, which can be modified in two days by one person." From another aspect, assume that this concrete scenario is used to specify a quality attribute of an architecture and that architecture applies the Front Controller pattern, since this concrete scenario is an instance of a general scenario extracted from the Front Controller pattern used in that architecture, this increases confidence in that architecture's ability to satisfy the above-described concrete scenario [9]. Thus, compared with standard pattern documentation formats, the *ASIP* captured in the *ASIP* template not only presents architecturally significant information at an appropriate level of abstraction for architecting activities, it helps improve architecting process in several ways. For example,

1. It can help architects/designers identify patterns used in a software architecture and understand the reasons of using those patterns. Moreover, it can also help architects/designers identify appropriate

patterns for system design. Additionally, by looking at the scenarios, quality attributes, and tactics supported by the patterns, a software architect can be confident that architecture is capable of supporting specific general scenarios.

2. The *ASIP* can improve the scenario development workshops for architecture evaluation process by helping stakeholders to develop concrete scenarios for an application based on the general scenarios extracted from the patterns used in the architecture of that application. In addition, general scenarios extracted from patterns can stimulate stakeholders' thinking resulting in improved quality of scenarios.

5. DESCRIPTION OF THE EXPERIMENT

5.1 Research hypotheses

Informally, our experimental hypothesis is that the *ASIP* makes the pattern-based architecturally significant knowledge available to architects and designers in a succinct format that is more helpful in gaining a high level understanding of a pattern's objectives and rationale than reading that pattern's standard documentation. The contexts for the experiments described in this paper are architecture comprehension and design tasks where architecture/designer either understands an existing architecture before making appropriate modifications or design a new architecture to achieve desired level of quality attributes. Our study involves two experiments involving the same subjects where the first experiment investigated a comprehension task and the second experiment investigated a design task.

More formally, the null hypothesis for our comprehension experiment (experiment 1) is:

H01: Pattern-based architecture knowledge presented as *ASIP* is no more useful in identifying and understanding patterns used in software architecture of an application than pattern documentation itself.

Our alternative hypothesis is:

H1: Pattern-based architecture knowledge presented as *ASIP* is more useful in identifying and understanding patterns used in software architecture of an application than pattern documentation itself.

For the design experiment (experiment 2) our null hypothesis is

H02: Pattern-based architecture knowledge presented as *ASIP* is no more useful in selecting and justifying patterns to be used in software architecture of an application than pattern documentation itself.

Our alternative hypothesis is:

H2: Pattern-based architecture knowledge presented as *ASIP* is more useful in selecting and justifying patterns to be used in software architecture of an application than pattern documentation itself.

Experiment tasks	Treatment	ASIP provided	Pattern documentation provided
	Material		
Understand a pattern-based architecture	E-commerce application	Group A	Group B
Design a pattern-based architecture	Client relationship management application	Group B	Group A

FIGURE 2 DESIGN OF EXPERIMENTAL CONDITIONS

5.2 Experiment Design

The experiment design was a balanced randomized design i.e. the same number of subjects was assigned to each experimental condition [20], with a treatment group (who used *ASIP* templates to support their experimental task) and control group (who used standard pattern documentation to support their experimental task). At the beginning of the study, subjects were assigned at random to two groups (group A and Group B), subjects remained in the same group for each experiment. In the first experiment group A was assigned to the treatment condition and group B was assigned to the control, condition. In the second experiment, group A was assigned to the control condition and group B to the treatment condition. The allocation of groups to treatment and control for experiment 1 was randomized.

The **Independent variable** manipulated by experiment 1 is the support information provided to the participants, either *ASIP* (treatment) or standard J2EE pattern documentation (control).

The **Dependent variable** is score each participant achieved for the comprehension task based on the number of correctly identified and explained patterns used in the given architecture.

The **Independent variable** manipulated by experiment 2 is again the support information provided to the participants, either *ASIP* (treatment) or standard J2EE pattern documentation (control).

The **Dependent variable** is score each participant obtained for the design task based on the number of patterns correctly selected and justified to satisfy desired quality attributes.

5.3 Participants

The 20 participants in the study were postgraduate students of software architecture course offered at the University of New South Wales, Australia. Since this study was incorporated in the syllabus of the course, most of the tasks of the study were part of the course assessment. The students were briefed about the objectives and procedures of the study. They had the option of withholding their results from the study. Written permission was sought from the student to use their data in this study.

The ratio of male and female was representative of the traditional software engineering courses and industry with only 5 female students. All of the participants were either working or had worked as information technology (IT) professionals with an average working experience of 4.5 years in the IT industry and an average age of 27 years. Their working experience typically had a good mix of design, coding, test, maintenance, and technical support activities.

TABLE 2 PARTICIPANTS' LEVEL OF EXPERTISE IN DIFFERENT SOFTWARE ARCHITECTURE RELATED KNOWLEDGE

Level of expertise	SA Design	SA Styles & patterns	Design patterns	J2EE patterns
None	0	2	0	0
Novice	6	10	8	4
Skilled	11	6	10	16
Expert	3	2	2	0

Participants had varying level of expertise in architecture design, architecture styles or patterns, design patterns, and J2EE design patterns as shown in Table 2. These data were collected from the participants on the first day of the course.

5.4 Training

All participants had a good understanding of the principles of architecture design and evaluation, and patterns, in particular J2EE design patterns both as a result of their academic background, work experience and the training they received during the course. All the participants had been involved in application design during their professional life, and most of them professed to have good knowledge of design principles and well-known design patterns.

The class exercises and assessment tasks in the course were designed around J2EE framework [18]. Prior to participating in the experiments, the participants were extensively exposed to the use of J2EE core design patterns by performing several tasks. For example,

- They had been introduced to the concept of pattern-mining for architecturally significant information and they undertook class exercises to give them experience of the information extraction and documentation process [9].
- They had participated in a workshop to develop scenarios for specifying quality requirements for two web-based applications.
- They had completed a major assignment to design and evaluate software architecture for an E-Commerce application using J2EE core design patterns [18], observing architecture design principles described in [1], and applying architecture evaluation methods presented in [21].

5.5 Experimental Material and Instruments

5.5.1 Software requirements specification

This study used Software Requirements Specifications (SRS) for two different applications, an e-commerce application, which one of the researchers had developed for a commercial organization, called Water Hobbies Online (WHO) for confidentiality reasons, and a contact information management (CIM) component of a customer relationship management application described in [22]. Architecturally WHO is akin to JavaPet Store, a sample application used in several tutorials on designing enterprise applications for the J2EE platform. Both applications were designed using J2EE MVC architecture framework and J2EE core design patterns.

An important reason of using the SRS for these applications was to ensure that the participants would not have prior knowledge of the applications. Architectural aspects of both applications were not in the public domain when the studies were conducted.

We prepared simplified versions of SRS and descriptions of these systems along with structural view of architecture of the WHO online and some screen shots of the CIM component to provide the participants as clear a picture of the systems as possible. Apart from the functional requirements for each system, the SRS also included the general scenarios to specify the quality requirements and use case models of each application.

5.5.2 Patterns used

During the course, we demonstrated the process of extracting and documenting the architecturally significant information from patterns by mining several patterns described at different level of abstraction, e.g. architectural patterns [2, 15], design patterns [15], and framework-based patterns. We decided to provide the treatment group with the ASIP extracted from the patterns used in the Hobbies online application as it was developed on a J2EE platform and its architecture uses a number of J2EE patterns, such as MVC, View Helper, Front Controller,

Session Façade, Data Access Object, etc. For both experimental tasks, the information provided to the treatment group was extracted from the core J2EE patterns [18]. The information was extracted and documented using the template designed to represent architectural design knowledge (see Table 1).

5.5.3 Experimental tasks and marking scheme

In order to evaluate the value of pattern-based design knowledge captured as ASIP compared to standard pattern documentation, we developed a marking scheme for each of the experiments depending on the nature of the experimental task.

Experiment 1 required the participants to identify architectural pattern and design patterns used in the architecture of the WHO. They were also required to provide a brief description of the rationale for using each of the identified patterns and quality attributes affected (positively or negatively), plus at least one general scenario that the identified pattern could have supported in that architecture. According to the marking scheme, each correctly identified pattern received 1 mark, 2 marks were given for correctly describing the rationale, and each correctly reported quality attributed affected (positively or negatively) by one particular pattern received one mark, and a general scenario received one mark as well. For example, an answer in the following format will receive full marks for correctly identify a pattern used in the architecture. The pattern identified is Front Controller (see Table 1):

Pattern Id on the diagram: 4

Pattern name: *Front Controller*

Rationale: *This pattern provides a centralized access point for handling requests for multiple services from a user. It handles requests and manages content retrieval, security, and navigation, delegating to a Dispatcher component to dispatch to a View. A centralized point of contact for handling a request is usually considered a useful strategy.*

Quality attributes affected: *Reusability, Manageability, centralized services (Positive), Efficiency, Level of indirection (Negative).*

Supported scenarios: *A user wants to make multiple purchases in one session with the option of adding and deleting items from a shopping cart.*

If someone correctly provided information in this manner, he/she would get 1 mark for the pattern, 2 marks for the rationale, 1 mark for each quality attribute mentioned (positively or negatively affected). That means the above answer would receive a total of 9 marks. There was no negative marking. However, an incorrect identification number or name of the pattern means no mark for that particular pattern.

Experiment 2 required the participants to determine a suitable set of design patterns and one architecture pattern that can satisfy the quality requirements for the CIM. They were also required to provide the rationale, names of quality attributes affected (positively or negatively), and one scenario supported by each of the selected pattern. According to the marking scheme for this task, each correctly identified pattern in the proposed architecture received 1 mark, 2 marks were given for providing correct rationale, 1 mark for each correctly reported quality attributed affected (positively or negatively) by that particular pattern, and a general scenario received one mark. For example, an answer in the following format would have received full marks for proposing an architecture that uses Data Access Pattern.

Pattern name: *Data Access Object*

Rationale: *This pattern provides loose coupling between the business and resource tiers. Components need to be transparent to the actual data source implementation to provide easy migration to different vendor products.*

Quality attributes affected: *Portability, Transparency (Positive), Complexity, More layers (Negative).*

Supported scenarios: *Application can easily be portable to different data storage system.*

Again if the information provide for each part of the answer is correct, there would be a maximum of 8 marks for the Data Access Object pattern.

Application of this marking scheme may be affected by subjective opinion of the makers with regard to the rationale and scenario (pattern name and quality attribute affected were determined from pattern documentation). To avoid bias, two researchers mark the answers. Disagreements were discussed and resolved before assigning final marks to each answer. Moreover, this marking scheme is similar to the approaches used in [23] for counting the degree of requirements fulfilment within subsets of a programming maintenance task and in [24] for counting the correctly identified responsibilities of a cancel command to assess the quality of the participants' solutions.

5.5.4 Post-experiment Questionnaire

Each participant completed a questionnaire at the end of the study. The questionnaire asked the participants to give their opinion of the usefulness of the ASIP for understanding or designing architecture. This question required the participants to respond by circling a choice on a five point ordinal scale and providing a short explanation of their respective choice.

5.6 Experimental Validity

5.6.1 Threats to internal validity

Internal validity is the degree to which the values of dependent variables can only be attributed to the experimental variables[20].

In order to avoid bias in allocating participants to the treatment groups, we randomized the assignment by using card sort method. We wrote the names of the participants and groups on plain cards. After shuffling the cards, we assigned one card to each group without seeing the individual's or group's name on the card.

Another threat to the internal validity of our experiment is the approach used to mark the outputs of the experiments. One potential threat associated with the marking scheme is that part of the answer is marked based on the subjective assessment of the marker regarding the scenarios supported by a pattern and the reasons provided to use that pattern. We addressed this issue by having two researchers independently mark the experiments' outputs. Both markers discussed and resolved the differences before finalizing the final score for each participant's output in each experimental task.

A threat to the validity of experiment 2 is that it was not a completely independent experiment because the subjects were the same as those who took part in experiment 1. This means the results of experiment 2 might have been affected by the preceding experiment and the learning experience it presented to the participants. We would expect this to reduce the difference between treatment and control in experiment 2, so any bias should be against rejecting the null hypothesis.

The major threat to internal validity is the issue of experimenter and subject expectations. We were evaluating a technique we ourselves proposed and our enthusiasm for the technique may have influenced the participants and/or affected the way in which we marked the papers. This issue can only be addressed if other independent researches are prepared to replicate our experiments.

5.6.2 Threats to external validity

External validity is the degree to which the results can be generalized, i.e. transferable to other similar situations [20]. In particular, it is important to consider whether the participants are representative of the architects and designers who would design and maintain architecture in industry, and whether the experimental materials and process are representatives of the process and materials used in industrial architecture design.

Architects and designers in an industrial situation are more likely to have considerable experience and knowledge in architecture design and evaluation using design principles and patterns. That means our results are most likely to apply to IT professionals who have around 5 years of experience in software design and development. The requirement specifications used in the experiment were simplified versions of two real systems. The full systems had a large set of functional and non-functional requirements, which may require hundreds of large and small architectural decisions and consideration of applying patterns at various levels of abstraction. However, in industry, designers usually have a much longer time period in which to understand problem, identify candidate patterns to address the problem and assessing their viability.

5.7 Experimental procedure

The experiments were conducted in November 2004 as an assessment tasks in a software architecture course as mentioned in section 5.4. All the 20 participants arrived according to the experimental schedule. We had already designed the seating arrangement based on the random assignment of each participant to one of the two groups (control and treatment). For the first experimental task, we also had the description of the problem (SRS for WHO system), explanation of the task to be performed, desired format of the answer, and support material (pattern documentation (control group), and the ASIP (treatment group) placed according to the sitting arrangement. However, participants were not allowed to read any of that material until they were briefed on the process and protocols of both experimental tasks. After the briefing, the participants were asked to start working on the first task (as described in Section 5.5.3) and they were given 40 minutes to complete the tasks. Once 40 minutes elapsed, participants submitted their answers.

There was a short break between the two experimental tasks. During the break, the research team placed the second set of documents (SRS for CIM, task explanation, support material (the ASIP (treatment group) and pattern documentation (control group)) according to the experimental design - for the second experimental task, participants were assigned to different groups (control and treatment). Those who had already worked in the control group were placed into the treatment group and vice versa. Once the break time finished, participants returned to their seats. They were briefed on the task and instructions to be followed. The second experimental task was to identify suitable design patterns and architecture framework that should be incorporated in the architecture of CIM in order to achieve desired quality requirements (see Section 5.5.3 for details). They were given 50 minutes for this task. Once 50 minutes elapsed, solutions were collected. Participants filled a post-session questionnaire after which there was a debriefing session.

5.8 Data Collection

We collected three sets of data during the experiment: J2EE design patterns identified in a given architecture of a system, J2EE design patterns selected to be used to design architecture for satisfying a given set of quality requirements, and the questionnaire filled by all the participants to provide their opinion about the usefulness of the *ASIP* compared with pattern documentation. The participants were required to provide their answers for both tasks in a format described in the section 5.5.3.

6. RESULTS AND ANALYSIS

6.1 Statistical Results

The results of the descriptive statistics analysis of the raw scores for solutions to experimental task 1 are presented in Table 3 and Figure 3.

TABLE 3: SUMMARY STATISTICS FOR ARCHITECTURE COMPREHENSION TASK

	Participants	Mean	Standard Deviation	Standard Error
ASIP provided	10	23.90	12.71	4.02
Pattern documentation provided	10	13.80	11.12	3.52

FIGURE 3: BOX PLOT OF SCORES FOR PARTICIPANTS IN THE TREATMENT AND CONTROL GROUP

Since the box plot in Figure 3 showed an outlier in the control group, we performed a non-parametric one-tailed test on the ranked data (i.e. the Mann-Whitney test). The results indicated that the average score for the treatment group was significantly larger than the average score for the control group ($p=0.0375$). The box plot in Figure 3 shows one outlier in the control group.

The results of the descriptive statistics analysis on the raw scores for solutions to experimental task 2 are presented in Table 4 and Figure 4.

TABLE 4: SUMMARY STATISTICS FOR ARCHITECTURE DESIGN TASK

	Participants	Mean	Standard Deviation	Standard Error
ASIP provided	10	26.85	9.49	3
Pattern documentation provided	10	19.60	7.25	2.3

Using Mann-Whitney test, the results indicated that the average score for the treatment group was significantly larger than the average score for the control group ($p=0.035$).

FIGURE 4: BOX PLOT OF SCORES FOR PARTICIPANTS IN THE TREATMENT AND CONTROL GROUP

6.2 Analysis of the Self-reported Data

In addition to collecting objective quantitative data as part of the two experiments, we also used a three-part questionnaire to obtain self-reported data. The self-reported information was subjective but, for the most part, quantitative (i.e. assessed on a nominal scale at worst). We also obtained more qualitative data from the subjects in response to open-ended questions. Following is the analysis of the two questions of the questionnaire. The first question was asked after the first part of this research program (an observation study, see [9] for details) and the second question was asked after the experiment reported here.

Predicted value of the ASIP. When asked how useful the ASIP would be during architecture design and evaluation [9], all the respondents seemed convinced of the value of the ASIP during architecture design and evaluation activities. 11 (61%) thought it would be useful, 6 (33%) said very useful and one person (6%) thought extremely useful. One of the respondents commented on usefulness of ASIP for architecting activities in these words *“Extracted information presented in templates illustrate clearly the advantages and disadvantages of using a pattern, and provides an easy approach to choose specific pattern for specific requirements”*. We consider the responses to this question to correspond to predicted value because at that time the respondents had not used the ASIP for any of the architecting task. However, the responses to the next question confirmed their predictions.

Usefulness of the ASIP for pattern-based design. Having performed both of the experimental tasks, all of the participants of the study had been provided with the ASIP as support material either for the first task or the second task. The post-experiment study questionnaire asked them which type of pattern-based architecture knowledge was more helpful in performing the tasks. 18 (90%) of the respondents believed the ASIP was more help than standard pattern documentation. Two (8%) thought otherwise. Most of the respondents found the ASIP documented in template more precise and to-the-point. They commented that it was easy to read and understand and could be used as a quick reference during architecture design process. For example, one of the responses was *“Architectural information presented using template seems more integrated and informative”*.

7. DISCUSSION AND CONCLUSION

The results of the analysis of quantitative data obtained by marking the solutions to architecture comprehension and architecture design tasks demonstrate significant effect of providing pattern-based architecturally significant information as the ASIP compared with the standard pattern documentation. The treatment groups (i.e., participants provided with the ASIP) performed significantly better than the control groups (i.e., participants provided with the standard pattern documentation) on both architecture comprehension and architecture design tasks. The participants who received the ASIP obtained, on average, 10 marks more (23.90 compared with 13.80) on the task of identifying and understanding the use of certain patterns in a given architecture (comprehension task) than the participants who were provided with the standard pattern documentation. While for the task of selecting suitable patterns for designing architecture, the participants who were given the ASIP obtained, on average, 7 marks more (26.85 compared with 19.60) than the participants who were provided with the standard pattern documentation.

The results of Man-Whitney statistical testing to evaluate the hypotheses for both experimental tasks revealed that the differences between the treatment and control groups for the first and second tasks were significant ($p=0.0375$ and $p=0.035$ respectively). The findings from the statistical analysis indicate that the ASIP is more helpful in tasks involving identifying and understanding patterns used in a software architecture and tasks involving selecting appropriate patterns to design a software architecture compared with standard pattern documentation.

Apart from obtaining the objective quantitative data based on the solutions to the experimental task to assess the value of the ASIP in software architecting process, we also gathered self-reported data. We believe that subjective information provides insights into how acceptable methodological changes and innovations will be to the people who will be expected to employ them. Moreover, this type of information is essential if process change is to be successful. For obtaining self-reported data, we used a three part questionnaire throughout this research program (one observational study, two controlled experiments), one part of questionnaire was administered after each part of the study. The analysis of the one of the questions of the first part of the questionnaire administered after the first study [9] revealed that the participants of the study were convinced of the value of the ASIP during software architecture design and evaluation even before using the ASIP for any of these tasks. After using the ASIP for architecture comprehension and design tasks in this experiment, the majority of the participants believed that the ASIP was more useful in both tasks than the standard pattern documentation.

One of our main research goals is to improve the architecting processes by gathering and documenting architecturally significant information in a format that is readily useful in making design decisions with an informed knowledge of the consequences of those decisions. We aim to reduce the time, resources and skill level required to effectively and efficiently design and assess architectures with respect to desired quality attributes. We have proposed a technique of *“mining patterns”* as one of the steps towards that goal.

This paper reports one part of a research program designed to assess the value of architecturally significant information extracted from pattern (ASIP) to improve software architecture process. When we presented our initial work on capturing and documenting general scenarios by *“mining patterns”* for improving software architecture evaluation process [25], we also outlined some short term goals to extend and empirically validate our idea. One of those goals was to empirically demonstrate the value of pattern-mining exercise and of the extracted information for software architecting. In this paper, we present the design and results of the second of the two experiments of a research program aimed at achieving that goal.

Both experiments of this research program are aimed at gathering empirical evidence to support our view of the usefulness of the ASIP for improving architecting activities (e.g. developing quality sensitive scenarios and selecting appropriate patterns to design software architecture). The findings of the experiment reported in this paper provide empirical evidence to support the second of our claims that the ASIP is useful in providing high level understanding of a pattern's objectives and rationale for supporting software architecture comprehension and design tasks.

This paper concludes the reporting of the research program that we undertook to assess the different components of our pattern-mining framework and usefulness of the architecturally significant information distilled from pattern

and maintained as *ASIP*. We ourselves have evaluated our own research. Now we hope researchers and practitioners of architecture discipline will experiment with the research concept to further validate it. In particular, we hope others will replicate our experiment and contribute to a body of knowledge to determine the pros and cons of the “*ASIP*” to improve software architecture design and evaluation processes.

REFERENCES

- [1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. 2 ed. 2003: Addison-Wesley.
- [2] F. Buschmann, et al., *Pattern-Oriented Software Architecture: A System of Patterns*. 1996: John Wiley & Sons.
- [3] F. Bachmann, L. Bass, and M. Klein, *Deriving Architectural Tactics: A Step toward Methodical Architectural Design*, *Tech Report CMU/SEI-2003-TR-004*, SEI, Carnegie Mellon University, USA, 2003.
- [4] R. Kazman, S.J. Carriere, and S.G. Woods, *Toward a Discipline of Scenario-based Architectural Engineering*, *Annals of Software Engineering*, *Kluwer Academic Publishers*, 2000. **9**(1-4): pp. 5-33.
- [5] M. Ali-Babar, Scenarios, Quality Attributes, and Patterns: Capturing and Using their Synergistic Relationships for Product Line Architectures, *International Workshop on Adopting Product Line Software Engineering*, 2004.
- [6] D. Gross and E. Yu, *From Non-Functional Requirements to Design through Patterns*, *6th Int'l Workshop on Requirements Engineering Foundation for Software Quality*, 2000.
- [7] L. Bass and B.E. John, Linking usability to software architecture patterns through general scenarios, *Journal of Systems and Software*, 2003. **66**(3): pp. 187-197.
- [8] E. Folmer, J.v. Gorp, and J. Bosch, A Framework for Capturing the Relationship between Usability and Software Architecture, *Software Process Improvement and Practice*, 2003. **8**(2): pp. 67-87.
- [9] M. Ali-Babar, B. Kitchenham, P. Maheshwari, and R. Jeffery, Mining Patterns for Improving Architecting Activities - A Research Program and Preliminary Assessment, *Proceedings of the 9th International conference on Empirical Assessment in Software Engineering*, 2005.
- [10] M. Ali-Babar, B. Kitchenham, and P. Maheshwari, The Value of Architecturally Significant Information Extracted from Patterns: A Controlled Experiment, *Accepted in the Australian Software Engineering Conference*, 2006.
- [11] IEEE, IEEE Standard 1061-1992, Standard for Software Quality Metrics Methodology. 1992, New York: Institute of Electrical and Electronic Engineers.
- [12] S. Thiel, On the Definition of a Framework for an Architecting Process Supporting Product Family Development, *Proceedings of the 4th International Workshop on Product Family Engineering*, 2001.
- [13] M. Matinlassi, E. Niemela, and L. Dobrica, Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture, *Tech Report 456*, VTT Technical Research Centre of Finland, Espoo, 2002.
- [14] N. Lassing, D. Rijsenbrij, and H.v. Vliet, How Well can we Predict Changes at Architecture Design Time? *Journal of Systems and Software*, 2003. **65**(2): pp. 141-153.
- [15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns-Elements of Reusable Object-Oriented Software*. 1995, Reading, MA: Addison-Wesley.
- [16] K. Petersson, T. Persson, and B.I. Sanden, Software Architecture as a Combination of Patterns, *CrossTalk The Journal of Defense Software Engineering*, Oct., 2003.
- [17] E.L.A. Baniassad, G.C. Murphy, and C. Schwanninger, Design Pattern Rationale Graphs: Linking Design to Source, *International Conference on Software Engineering*, 2003.
- [18] D. Alur, J. Crupi, and D. Malks, *Core J2EE Patterns: Best Practices and Design Strategies*. 2nd ed. 2003: Sun Microsystems Press.
- [19] B.E. John, L. Bass, M.-I. Sanchez-Segura, and R.J. Adams, Bringing Usability Concerns to the Design of Software Architecture, *9th IFIP Working Conference on Engineering for Human-Computer Interaction*, 2004.
- [20] C. Wohlin, et al., *Experimentation in Software Engineering: An Introduction*. 2000: Kluwer Academic Publications.
- [21] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. 2002: Addison-Wesley.
- [22] L.A. Maciaszek and B.L. Liong, *Practical Software Engineering: A Case Study Approach*. 2005: Pearson, Addison-Wesley.
- [23] L. Prechelt, B. Unger, M. Philippsen, and W.F. Tichy, Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance, *IEEE Transactions on Software Engineering*, 2002. **28**(6): pp. 595-606.
- [24] E. Golden, B.E. John, and L. Bass, The Value of a Usability-Supporting Architectural Pattern in Software Architecture Design: A Controlled Experiment, *Proceedings of the 27th International Conference of Software Engineering*, 2005.
- [25] L. Zhu, M. Ali-Babar, and R. Jeffery, Mining Patterns to Support Software Architecture Evaluation, *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture*, 2004.