APEP – An Architectural Pattern Evaluation Process

Andreas Seitz, Technical University of Munich (TUM)
Felix Thiele, Technical University of Munich (TUM)
Bernd Bruegge, Technical University of Munich (TUM)

A pattern describes a solution to a recurring problem in a given context. When writing such a pattern, the author faces several challenges. The first writing attempts are often vague and not clearly formulated and do not sufficiently describe the pattern. Other software engineers do not understand the pattern well enough and cannot use it to solve their problems. In this paper, we introduce APEP, an **A**rchitectural **P**attern **E**valuation **P**rocess. It allows conducting a review of an architectural pattern draft based on adaptations of sophisticated methods for architectural evaluation. In this paper, we design and present the Review of Intermediate Architectural Patterns (RIAP) as a method. To prove feasibility and applicability, we conducted two case studies based on an exemplary architectural pattern.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures—*Patterns*

Additional Key Words and Phrases: Software Engineering, Patterns, Software Architecture, Architecture Patterns, Pattern Evaluation, Software Quality

ACM Reference Format:

Andreas Seitz, Felix Thiele, Bernd Bruegge. 2018. APEP – An Architectural Pattern Evaluation Process. 12th Latin American Conference on Pattern Languages of Programs (SLPLoP) (November 2018), 11 pages.

1. INTRODUCTION

A pattern is a solution to a recurring problem in a given context [Alexander 1977]. A reference model decomposes a problem into parts that in conjunction can solve the problem at hand. It always arises by experience and thus only appears in mature domains [Len et al. 2003]. In conjunction, both comprise a reference architecture.

A software architecture is an instantiation of a reference architecture. It describes the structure of the system, the relationship between its components and their externally visible properties [Len et al. 1998]. A software architecture is very expensive to change once it is implemented. Architectural evaluation is a method to identify issues before implementing an architecture to avert these costs [Clements et al. 2002].

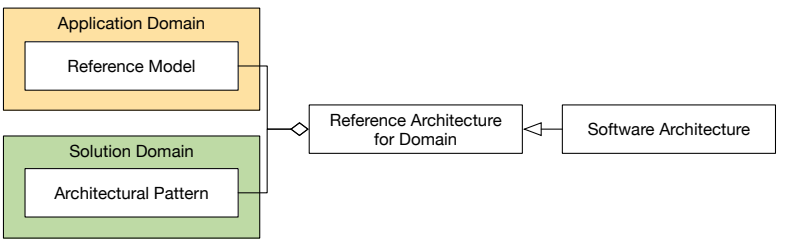

Fig. 1. Relationship between reference model, architectural pattern, reference architecture and software architecture as UML class diagram (based on [Len et al. 1998])

In contrast, no methods exist to evaluate an architectural pattern. We introduce an Architectural Pattern Evaluation Process (APEP) to guide an author when continuously evaluating an architectural pattern. It comprises multiple different evaluation methods. Those methods are based on existing evaluation methods for architectures and are specifically tailored to serve the purpose of pattern evaluation. In the course of this paper, we introduce these existing evaluation methods and

introduce APEP. Additionally, we contribute a tailored evaluation method for architectural patterns, a Review of Intermediate Architectural Patterns (RIAP). Finally, we present two case studies that show the feasibility of the presented evaluation tools.

2. FOUNDATIONS

When targeting a specific architecture, several alternatives exist on how to evaluate it. Abowd et al. divide them into two main categories: qualitative questions and quantitative measurements [Abowd et al. 1997]. Qualitative questions use scenarios or checklists, while quantitative measurements use metrics or simulations and experiments to assess the suitability of the architecture for a system in terms of requirements fulfillment. Scenario-based evaluation methods are for example the "Software Architecture Analysis Method" (SAAM) by [Kazman et al. 1994] and the "Architecture Trade-Off Analysis Method" (ATAM) by [Kazman et al. 2000]. A comparison of these methods can be found in [Babar et al. 2004] or [Dobrica and Niemela 2002].

While SAAM and ATAM target fully specified architectures, which develop over the course of a project, *Active Reviews for Intermediate Designs* (ARID) were introduced by Clements to be much more lightweight and concentrate on the viability of the software architecture [Clements 2000]. ARID is best suited to be used at the beginning of a design process to discover errors or inconsistencies and reduce the costs of such problems. In addition, it is more likely that the various stakeholders will accept the design if they are involved in advance.

ARID is a combination of two approaches: *Active Design Reviews* (ADR) and ATAM. ADRs were first introduced by [Parnas and Weiss 1985]. It is a method that actively challenges reviewers to solve review tasks using the design in relevant scenarios instead of asking yes/no questions (cf. Table I based on [Clements et al. 2002]).

Table I. Conventional design review question vs. active design review instruction

Conventional Design Review	**Active Design Review**
Are exceptions defined for every program?	Write down the exceptions that can occur for every program.
Are the right exceptions defined for every program?	Write down the range or set of legal values of each parameter. Write down the states under which it is illegal to invoke the program.
Are the programs sufficient?	Write a short pseudo-code program that uses the design to accomplish [some defined task].

The general outline of ARID is based on the structure of ATAM. It is divided into two phases: the rehearsal and the review. In the rehearsal, the lead designer and a facilitator meet up to create the exercises for the review. This phase is comprised of the following four steps:

(1) **Identify the reviewers:** As ARID evaluates an intermediate architecture, the reviewers best suited for the job are the software engineers that are expected to apply the architecture. In this step, a group of around a dozen software engineers is selected.

(2) **Prepare the design briefing:** The lead designer is supposed to introduce the design to the reviewers. This step is used to prepare the corresponding presentation and to do a dry run. The dry run is presented to the facilitator, which brings benefits: first, the facilitator might come up with questions for which the designer can prepare. Second, the presentation itself is evaluated both content-wise as well as from a presentation perspective. And finally, the designer can practice the presentation to give it in the given time frame. The presentation should not exceed two hours.

(3) **Prepare the seed scenarios:** The lead designer and facilitator then continue to craft several scenarios, in which the design could be used. These do not need to be used for later evaluation but can serve as a valuable starting point and generally make the design more understandable.

(4) **Prepare the materials:** Finally, the rehearsal concludes by preparing all necessary materials for the review, like copies of the presentation, seed scenarios and agenda.

In the review phase, the stakeholders conduct the review by following the five steps:

(5) **Present ARID:** The facilitator starts the review by presenting ARID and the steps involved.

(6) **Present the design:** The lead designer presents the design briefing and introduces the seed scenarios. While doing so, the audience is only allowed to ask comprehension questions; questions regarding rationale or suggestions are not allowed. All comprehension questions are noted down by a scribe; those questions indicate a lack of clarity in the design or its documentation.

(7) **Brainstorm and prioritize scenarios:** In the next step, the group brainstorms scenarios in which the design could be used. The seed scenarios are included in the pool. Each participant gets a vote total of 30 percent of the number of scenarios. They are free to allocate all their votes to one specific scenario that they deem important or distribute them between the different scenarios as they wish. The scenarios that receive the most votes are used in the review.

(8) **Apply the scenarios:** In this step, the facilitator asks the group to craft real or pseudo code using the design to tackle the problem presented in the scenarios. Usually, the group starts with the scenario that received the most votes. In some cases, it might be more useful to start with an easier one, if the top-rated scenario seems to be too daunting of a task. During this step, the lead designer is not allowed to help the group or give hints. Only when the group becomes stuck the facilitator is asked to intervene, and the lead designer may steer the group in the right direction. Every time this happens, an issue is recorded, which indicates a flaw in the design. The reviewers are also asked to bring up any discrepancies they uncover as issues. This step continues until either time has run out, the top-rated scenarios are solved or the group concludes that the design is suitable or unsuitable.

(9) **Summarize:** Finally, the facilitator recalls all identified issues, thanks the reviewers for participating and asks for feedback regarding the review.

The result of ARID, SAAM and ATAM is a list of issues. This list contains issues that the reviewers have uncovered in the architecture, e.g. that certain quality features cannot be met or that the design is not feasible. These issues help the designer improve the architecture.

3. APEP

Writing an architectural pattern is an iterative process [Wellhausen and Fießer 2012]. The goal of APEP is to support a pattern designer in evaluating and iteratively improving an architectural pattern and its formulation. The process begins and ends with the architectural pattern and can be applied as soon as the first draft of a pattern is written or at a later point in time. An evaluation method is chosen and conducted. Based on the resulting issues, the pattern is improved and another iteration of APEP can be started (cf. Figure 2).

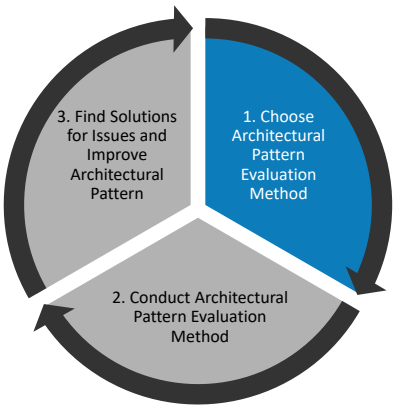

Fig. 2. Overview of the iterative APEP process

3.1 Design

APEP requires two roles: designer and reviewer. The designer provides the architectural pattern and is responsible for introducing it and answering questions that arise during the review. The facilitator's role is to guide through the process and capture issues.

As a prerequisite for APEP, the designer must describe an architectural pattern. APEP can be used both for the early evaluation of non-mature patterns - which we call Intermediate Architectural Patterns - and for the evaluation of mature patterns. Figure 3 presents an overview of the different

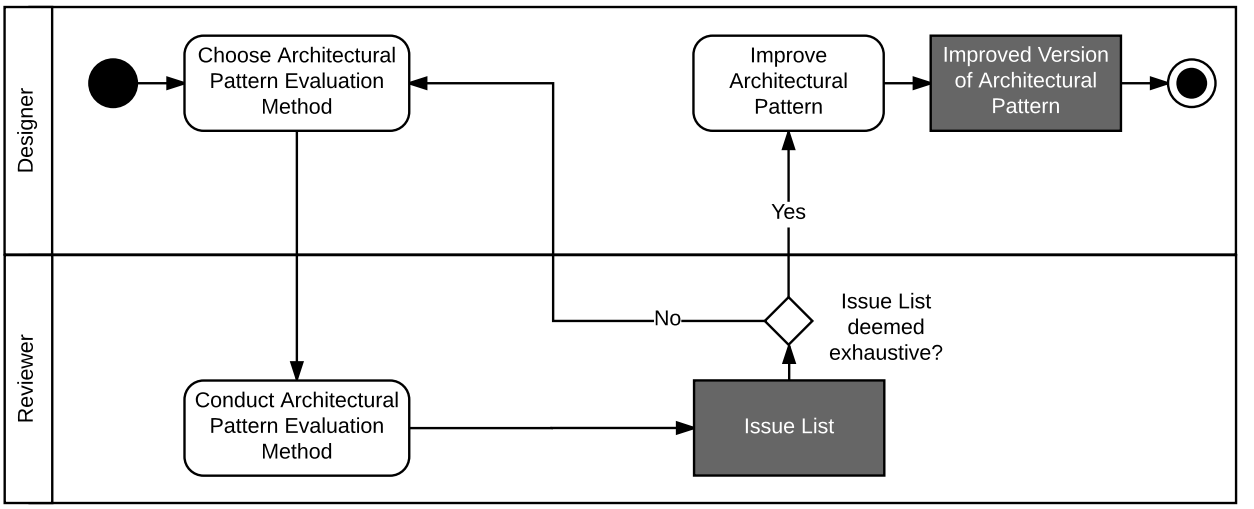

Fig. 3. Overview of one iteration of APEP as UML activity diagram

steps and artifacts of APEP. The designer initiates the process by choosing an architectural pattern evaluation method. These methods are delineated in Section 3.2. The reviewers conduct the architectural pattern evaluation method and bring up issues. Those issues are gathered in an issue list and described further in Section 3.3. When the issue list is filled, the designer can work on an improved architectural pattern. Otherwise, another architectural pattern evaluation method is chosen and conducted.

3.2 Architectural Pattern Evaluation Methods

We differentiate evaluation methods for architectures and architectural patterns:

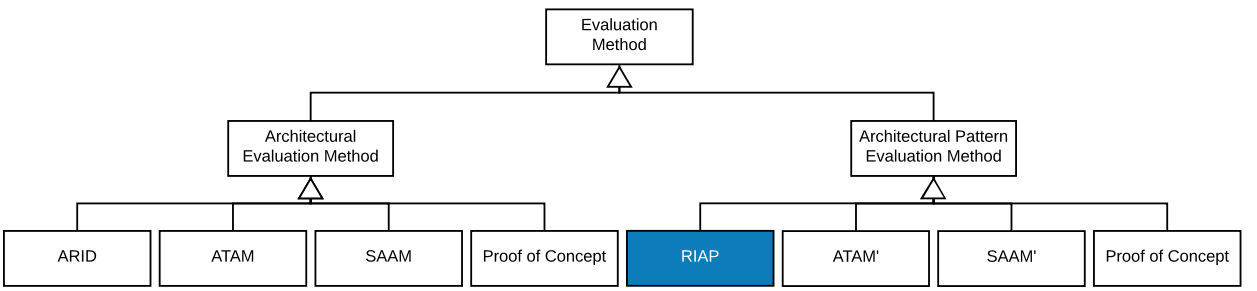

Fig. 4. Taxonomy of methods for architectural evaluation and architectural pattern evaluation as UML class diagram

Based on the methods for architectural evaluation depicted in Section 2, we propose to adapt them for architectural pattern evaluation. We present an adaption of ARID — namely RIAP, a **R**eview for **I**ntermediate **A**rchitectural **P**atterns. ATAM' and SAAM' are adapted versions of ATAM and SAAM. A proof of concept implementation shows both the feasibility of an architecture and an architectural pattern through the implementation of a specific scenario.

The architectural pattern evaluation methods enable the reviewers to instantiate the pattern, elicit quality requirements and assess the resulting architecture against those requirements. The sequence of activities for conducting the architectural pattern evaluation method is shown in Figure 5.

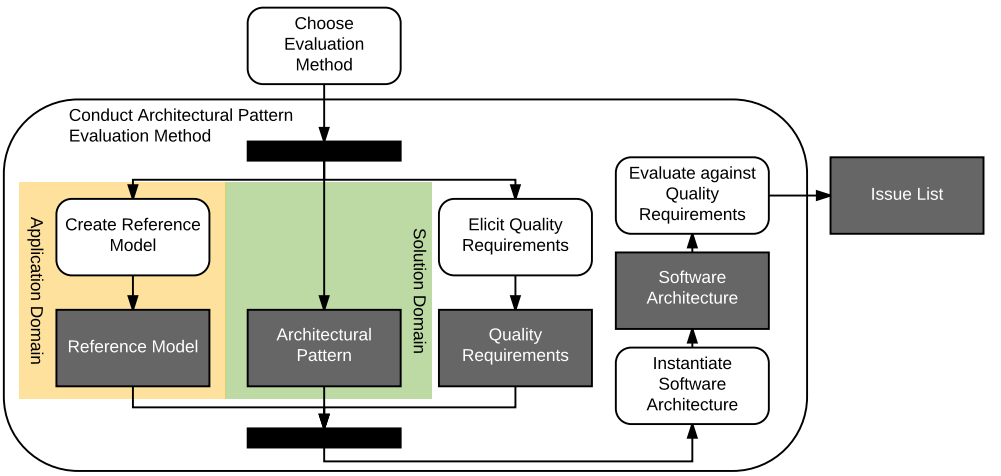

Fig. 5. Conduct architectural pattern evaluation method in detail as UML activity diagram

With respect to Figure 1, a software architecture is the result of a reference model and architectural patterns. Therefore, the evaluation method requires the reviewers to create a reference model. The architectural pattern is provided by the designer. In addition, quality requirements must be defined to assess the suitability of the architecture. With those three artifacts, the reference model, the architectural pattern and the quality requirements, the reviewers can instantiate a software architecture. This is evaluated against the elicited quality requirements and results in a list of issues, which contains all issues that were uncovered during the process and the evaluation against the quality requirements.

3.3 Issues

All described architectural pattern evaluation methods bring up *Issues*. These issues arise in a *Context*, for example during the implementation of the communication between two components of the architectural pattern. They feature a *Problematic Solution* and give rise to an *Improved Solution*. The improved solution is used to enhance the architectural pattern.

This tripartite is used in the style of an anti pattern as described in [Brown et al. 1998]. The *Context* is adopted directly. We merge *Problem* and *Problematic Solution* into the problem of the issue and the *Refactored Solution* transitions to our *Solution*.

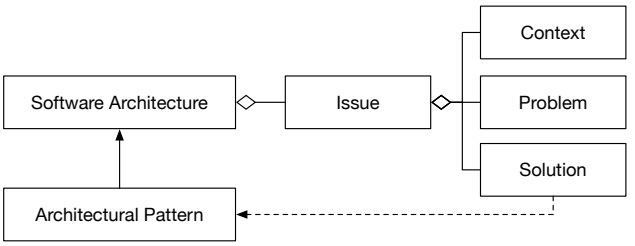

Fig. 6. Structure of the *Issue* in context of APEP as UML class diagram

4. RIAP

As an architectural pattern evaluation method for APEP, we present RIAP - a **R**eview for **I**ntermediate **A**rchitectural **P**atterns. RIAP is an adaption of ARID (cf. Section 2) for architectural patterns. Like ARID, RIAP is capable of early evaluations of intermediate architectural patterns.

4.1 Design Goals

The goal of RIAP is to strengthen the evaluated pattern. RIAP is part of the APEP process and is used for the early evaluation of intermediate architectural patterns.

4.2 Characteristics

During the design of RIAP, we considered the applicability in practice. In the sense of requirements, we distinguish between functional and non-functional characteristics: features that RIAP must fulfill (functional characteristics) or a restriction it must obey (non-functional characteristics) to be able to carry out the specified objective of strengthening the pattern [Brügge and Dutoit 2004]. In consultation with other pattern authors and inspired by other methods of architecture evaluation methods, we identified the following functional and non-functional characteristics for RIAP:

CR1 - Identify Issues: RIAP must identify issues of the architectural pattern.

CR2 - Improve Pattern: The identified issues must be usable to improve the pattern in terms of context, problem or solution description. There may also be negative consequences that need to be considered.

CR3 - Instantiate Architecture: According to APEP, an evaluation method must instantiate a concrete architecture. RIAP must instantiate an architecture that includes the architectural pattern that is evaluated.

CR4 - Provide Guidance: In contrast to the underlying ARID method, RIAP targets an intermediates architectural patterns instead of an intermediate designs. This increases the scope of the review. RIAP must provide guidance during the review.

NCR1 - Time: RIAP must be able to be carried out in a reasonable time frame - we define reasonable in this context as less than four hours. In contrast to architecture evaluation methods described in Section 2, reviewers do not associate themselves with the pattern as much as with an architecture they would be using in a project.

NCR2 - Simplicity: Since RIAP targets intermediate architectural patterns, it must be feasible for undocumented patterns as well.

NCR3 - Heterogeneity of Participants: RIAP must be executable with a heterogeneous group of software engineers. Heterogeneous in this context means that they may not have experience in the domain the pattern is applicable, and the group may have different levels of experience.

4.3 Implementation

ARID serves as the basis for RIAP as it is lightweight, can be completed in a reasonable time frame and can also be used for intermediate architectural patterns. RIAP needs a facilitator, a designer and reviewers. The designer provides the architectural pattern and is responsible for introducing it and answering questions that arise during the review. The role of the facilitator is to guide the process and capture issues. The reviewers' task is to test and review the architectural pattern. They do not need experience in the domain where the pattern is applicable. RIAP is therefore applicable with a heterogeneous group of software engineers, although we expect knowledge of software architecture and software engineering in general. The duration of RIAP is set at three hours to stay within a reasonable time frame. The process is divided into two phases: the rehearsal and the review. In the rehearsal phase, the designer and facilitator prepare the actual review, which takes place in the review phase.

4.3.1 *Rehearsal.* The rehearsal consists of four different steps. We depict those steps in an activity diagram in Figure 7. The facilitator and the designer meet and select the reviewers first. The designer prepares a pattern briefing that explains the architectural pattern and chooses a problem domain in which to evaluate the pattern. Since an architectural pattern is used in a broader context than a design, we assume that the scenarios that the evaluators would develop were too different to evaluate. We therefore restrict all scenarios to a specified problem domain. The facilitator prepares the RIAP presentation that includes the design briefing and the description of the problem domain. We recommend printing both the design briefing and the problem domain on a handout that the reviewers can use throughout the review.

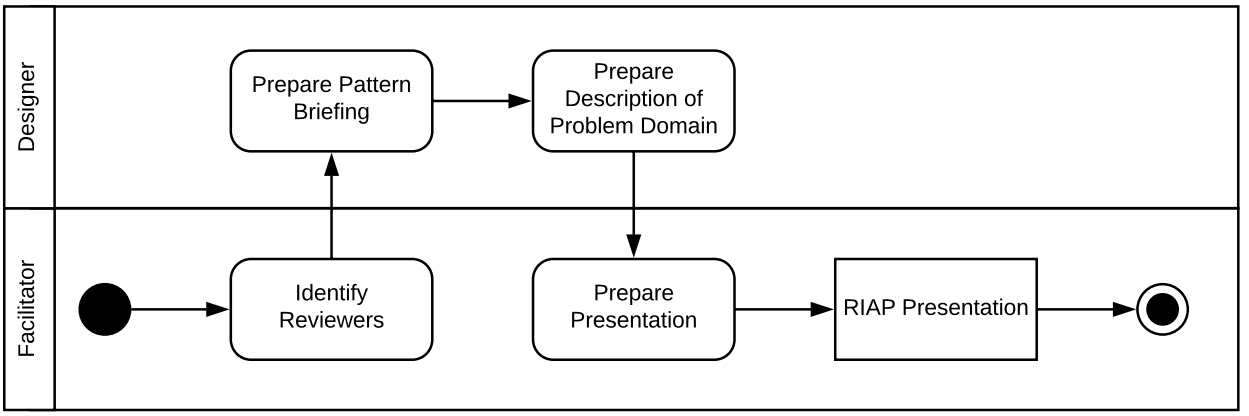

Fig. 7. RIAP rehearsal phase as UML activity diagram

4.3.2 *Review.* While ARID targets a design, RIAP targets an architectural pattern: evaluating this seems to be too daunting of a task for the reviewers to tackle on their own. RIAP therefore provides for strict guidance through the process. In the review phase (cf. Figure 8), the facilitator introduces RIAP to the reviewers by describing all the steps. The designer presents the architectural pattern. During this time, reviewers can ask questions about the understanding of the architectural pattern. These questions are recorded as issues by the facilitator. Afterwards, the designer presents the problem domain to the reviewers.

To guide the reviewers in the following more active part, we set a total of seven tasks. Each task is introduced by the facilitator. The reviewers complete the task, while the designer can answer questions. These questions target the understanding of the architectural pattern. The designer may not give his opinion on how to implement certain aspects of the system, as this would influence the reviewers design decisions. The questions as well as any problems that arise during the completion of the tasks, such as how to map some components of the pattern onto the scenario, are recorded by the facilitator. The seven tasks are described in the following:

T1: Brainstorming Scenarios. The goal is to generate scenarios in which the architectural pattern is applicable and which fit the proposed problem domain. We suggest using brainstorming techniques such as the gallery method[1] or brain sketching[2]. The reviewers write the scenario title onto a card and give a brief description. The cards are collected on a white-board and displayed visibly for all reviewers. The reviewers can ask questions, if a scenario is unclear. The process of writing cards and looking at the results can be repeated until the reviewers have run out of ideas.

T2: Prioritize Scenarios. Due to time constraints, it is not possible to implement every scenario developed by the reviewers. Therefore, the scenarios must be prioritized. Each reviewer has a total of three votes, which they can assign to the scenarios. We recommend giving the participants stickers and having them stuck to the cards on the white-board as described in the sticking dots method[3]. The top-rated scenario is used for all subsequent tasks.

T3: Formulate Demo Scenario. The scenario has a title and a brief description. To implement it, the reviewers formulate a detailed description of the flow of events (cf. [Brügge and Dutoit 2004]). The implementation must be able to carry out all these steps.

[1] see `https://www.mycoted.com/Gallery_method` (accessed: 08/13/2017)
[2] see `https://www.mycoted.com/BrainSketching` (accessed: 08/19/2017)
[3] see `https://www.mycoted.com/Sticking_Dots` (accessed: 08/19/2017)

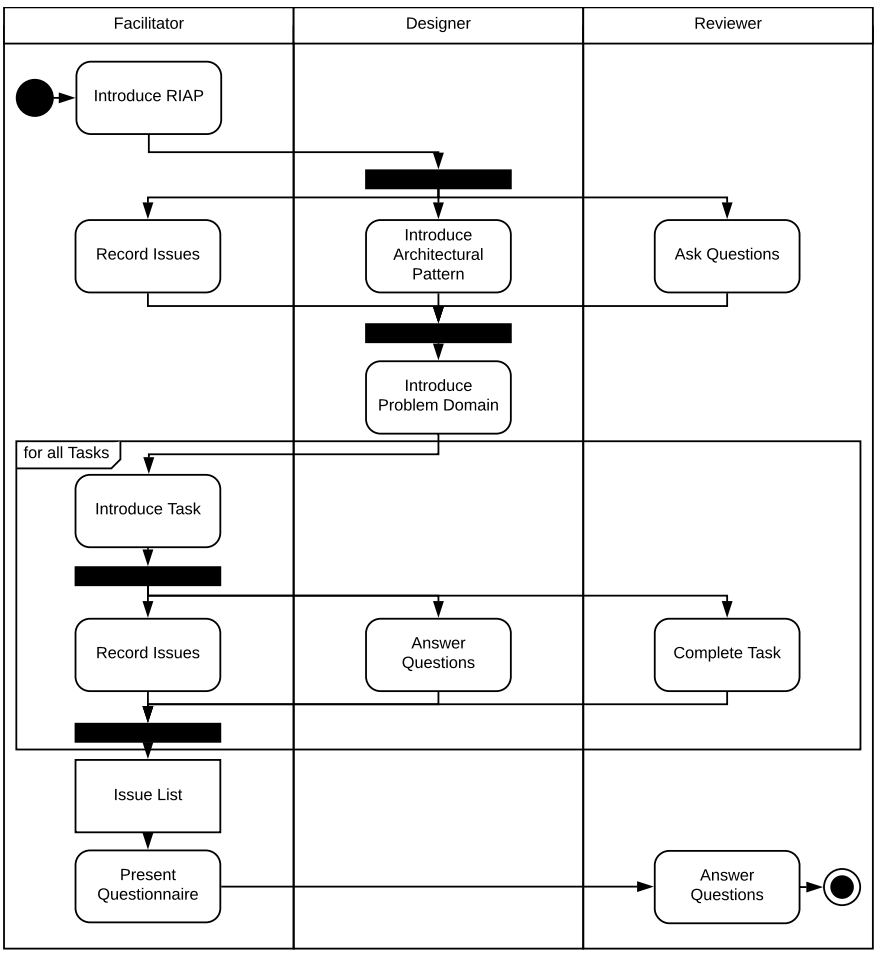

Fig. 8. RIAP review phase as UML activity diagram

T4: Elicit Non-Functional Requirements & Quality Requirements. The group elicits non-functional requirements that are important for the scenario and must be considered in the design to meet the quality requirements.

T5: Subsystem Decomposition. For designing the system, the reviewers create a subsystem decomposition (cf. [Brügge and Dutoit 2004]). All parts of the pattern must be included.

T6: Hardware/Software Mapping. To demonstrate that the provided subsystems can be mapped to hardware, the group creates a hardware/software mapping that shows the hardware used and the protocols used for communication between the different subsystems (cf. [Brügge and Dutoit 2004]).

T7: Apply the Demo Scenario. The group is supposed to show how the designed system carries out all individual steps of the demo scenario. For example, they can show the dynamic behavior of all participating subsystems. During this task, designer and facilitator can ask questions regarding the feasibility of the proposed solution. For example, they can examine how the system meets the non-functional requirements.

After all tasks have been completed, the facilitator creates a list containing all issues from the tasks and the initial presentation of the architectural pattern. To further encourage reviewers to provide feedback, RIAP concludes with a questionnaire. All questions are written in the style of active design review questions and aim to get reviewers to

question the architectural pattern in this scenario. The facilitator presents the questions about the architectural pattern — e.g. whether the pattern could meet all quality requirements, or more generally what the purpose of the pattern is — and the reviewers answer them.

5. CASE STUDY

To evaluate RIAP, we conducted two case studies in July 2017 based on an architectural pattern. A total of eight reviewers participated in the first case study and six participated in the second one. The first case study took three and a half hours while the second was reduced to three hours. The pattern to be evaluated is Fogxy, an architectural pattern for Fog Computing [Seitz et al. 2018]. The pattern was not yet published at that time and was still in a raw version. The results of RIAP were used to improve Fogxy.

5.1 Materials

To prepare for the case study, the designer and the facilitator met to craft a presentation and a handout[4]. The presentation contained the following slides:

- introduction to RIAP and the steps involved in the process
- design briefing for the architectural pattern
- problem domain
- task description
- questionnaire

5.2 Reviewers and Environment

We distinguish two groups of reviewers: software engineers familiar with the concepts of fog computing and people with less knowledge about fog computing. The first group consisted out of students researching in the area of fog computing. The second group were software engineering doctoral students. The review took place in a meeting room at university. The room offered magnetic boards for mounting or sticking cards and was large enough to work comfortably. An impression of the working environment is shown in Figure 9.

Fig. 9. Working environment and participants during the first RIAP meeting

[4]The presentation and handout of the case study are available at `https://github.com/andreasseitz/apep`

5.3 Problem Domain & Questionnaire

In the first case study, we used an example from the renewable energy domain. The second examined the architectural pattern in the field of autonomous driving. For the questionnaire, we decided to present the following active design review questions:

- What is the idea and purpose of the architectural pattern?
- What are the shortcomings in this scenario?
- Where did you have problems implementing it?
- Which non-functional requirements could not be fulfilled?

The goal of these questions was to allow the participants to revise the architectural pattern and scenarios and to give them the opportunity to address further issues.

5.4 Issue Example

As an example, we present an issue that raised in the course of the evaluation of Fogxy. The name of the issue is "Name Client Taxonomy" in the context "Understanding the Object Model of Fogxy". The reviewers described the problem that the class *Client* was too unspecific. The problem was written down by the facilitator as follows: "The taxonomy of the client is not clear in where to place e.g. smart textiles or a light switch." To solve this problem, the class *Client* was first renamed *Smart Object* and explained in the description of the pattern [Seitz et al. 2018].

5.5 Changes to the RIAP method

During the first case study, it became clear that our decision to select a problem domain and not provide seed scenarios was suboptimal. The tasks of brainstorming and prioritizing scenarios and determining non-functional requirements were not well received by the participants. These tasks were simply a means to an end — and thus took too long. Additionally, it was unclear to the reviewers how they were supposed to formulate scenarios independently of the architectural pattern evaluated. Therefore, we decided to predetermine the concrete scenario and its non-functional requirements. We ensure that the architectural pattern is generally plausible to be used in the scenario, and there is not much discussion about the different scenarios.

5.6 Findings & Limitations

Table II gives an overview and compares the two conducted case studies. Based on the introduced changes to RIAP, we were able to reduce the time of the evaluation to three hours. In both case studies, we implemented one scenario. In each case study, two groups formed, which presented different designs. The first case study discovered seven issues. The second case study verified these issues but did not bring up any new issues. This might be due to the chosen scenario. The questionnaire did not bring up any new insights in both sessions.

Table II. Metrics of the two RIAP case studies

	CS 1	CS 2
Time	3h 30min	3h
# reviewers	8	6
# created scenarios	8	1
# implemented scenarios	1	1
# different designs	2	2
# issues	7	6 (all duplicates)

6. CONCLUSION

In this paper, we introduce RIAP, a method to evaluate intermediate architectural patterns that is integrated in the APEP process — a process for architectural pattern evaluation. Two case studies showed that RIAP and APEP are valuable for improving an architectural pattern. The goal of APEP is to provide an iterative process for conducting an architectural pattern evaluation. During our research, we conducted two iterations: the first iteration using the RIAP method to identify issues that lead to an improved version of the architectural pattern; the second

iteration showing that this proposed improvement was feasible by implementing a proof of concept. The RIAP method was formatively evaluated during the case studies. We identified that the scenario creation took far too long in comparison to its results. We therefore defined the scenario for the RIAP method. The predetermination of scenarios improved the process by saving time and making the process clearer for participants.

In addition, we found that the questionnaire, unlike the tasks previously performed, did not yield the expected additional results. Some participants of the two groups were already familiar with the general concepts of the pattern and were able to express ambiguities in advance. For other participants, the questionnaire retains its justification by bringing the evaluation process to an orderly conclusion. APEP and RIAP should be applied to other architectural patterns to further refine both the process and the method. In our future work, we want to extend the investigation to patterns in general.

Acknowledgements

We thank our shepherd Antonio Maña for his valuable comments and feedback. Thanks to him we were able to improve the paper. We would also like to thank all participants of the case study and the writers' workshops.

REFERENCES

Gregory Abowd, Len Bass, Paul Clements, Rick Kazman, Linda Northrop, and Amy Zaremski. 1997. *Recommended Best Industrial Practice for Software Architecture Evaluation*. Technical Report. Software Engineering Institute, Carnegie Mellon University.

Christopher Alexander. 1977. *A pattern language: towns, buildings, construction*. Oxford University Press.

Muhammad Ali Babar, Liming Zhu, and Ross Jeffery. 2004. A framework for classifying and comparing software architecture evaluation methods. In *Australian Software Engineering Conference Proceedings*. IEEE, 309–318.

William J. Brown, Raphael C. Malveau, Hays W. McCormick III, and Thomas J. Mowbray. 1998. *Anti Patterns*. John Wiley and Sons, Inc.

Bernd Brügge and Allen H. Dutoit. 2004. *Object-oriented software engineering - using UML, patterns and Java*. Prentice Hall.

Paul Clements. 2000. *Active reviews for intermediate designs*. Technical Report. Software Engineering Institute, Carnegie Mellon University.

Paul Clements, Rick Kazman, and Mark Klein. 2002. *Evaluating Software Architectures*. Addison-Wesley.

Liliana Dobrica and Eila Niemela. 2002. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering* 28, 7 (2002).

Rick Kazman, Len Bass, Gregory Abowd, and Mike Webb. 1994. SAAM: A method for analyzing the properties of software architectures. In *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*. IEEE, 81–90.

Rick Kazman, Mark Klein, and Paul Clements. 2000. *ATAM: Method for architecture evaluation*. Technical Report. Software Engineering Institute, Carnegie Mellon University.

Bass Len, Clements Paul, and Kazman Rick. 1998. *Software architecture in practice*. Addison-Wesley.

Bass Len, Clements Paul, and Kazman Rick. 2003. *Software architecture in practice*. Vol. 2. Addison-Wesley.

David L. Parnas and David M. Weiss. 1985. Active Design Reviews: Principles and Practices. In *Proceedings of the 8th International Conference on Software Engineering*. IEEE Computer Society Press, 132–136.

Andreas Seitz, Felix Thiele, and Bernd Bruegge. 2018. Fogxy - An Architecural Pattern for Fog Computing. In *Proceedings of the 23Nd European Conference on Pattern Languages of Programs (EuroPLoP '18)*. ACM, New York, NY, USA.

Tim Wellhausen and Andreas Fießer. 2012. How to write a pattern?: A rough guide for first-time pattern authors. In *Proceedings of the 16th European Conference on Pattern Languages of Programs*. ACM, 5.

Received June 2018; revised November 2018; accepted December 2018