SPEM: A Software Pattern Evaluation Method

J. Kabbedijk, R. van Donselaar, S. Jansen
Department of Information and Computing Sciences
Utrecht University, The Netherlands
{J.Kabbedijk, R.VanDonselaar, Slinger.Jansen}@uu.nl

***Abstract*—Software architecture makes extensive use of many software patterns. The decision on which pattern to select is complex and architects struggle to make well-advised choices. Decisions are often solely made on the experience of one architect, lacking quantitative results to support the decision outcome. There is a need for a more structured evaluation of patterns, supporting adequate decision making. This paper proposes a Software Pattern Evaluation Method (SPEM), that enables the quantification of different pattern attributes by using structured focus groups. The method is formed using a design science approach in which an initial method was created using expert interviews, which was later refined using several evaluation sessions. SPEM helps software producing companies in structuring their decision making and selecting the most appropriate patterns. Also, SPEM helps in enriching pattern documentation by providing a way to add quantitative information to pattern descriptions.**

Keywords*—*architectural patterns. quality attributes. software architecture. decision making. pattern evaluation.

I. INTRODUCTION

Modern software architecture heavily relies on the use of many different software patterns, often used complementary to each other in order to solve complex architectural problems. Software architecture provides guidelines and tools for high level system design in which architects select best fitting patterns to be used within the software product [1]. Many different patterns and tactics exist, leading to a complicated trade-off analysis between different solutions and causing the evaluation and selection of the appropriate software patterns to be a complex task [2]. This complexity means architects need to have in-depth understanding of the project characteristics and requirements combined with extensive experience in software development.

The information needed for appropriate pattern selection is seldom available to all architects in a centralized or standardized way. Architectural decisions are frequently made based on experience and personal assessment of one person, instead of using the knowledge of many [3]. Allowing software architects to use all information efficiently saves time when selecting fitting software patterns and leads to better and more adequate decision making. For this to be possible, a method has to be created enabling the evaluation and documentation of crucial attributes of a software pattern [4]. This structured evaluation will allow architects and decision makers to compare different solutions and select the best matching pattern. Patterns, however, are a high-level solution that can be used different scenarios, making it impossible to use one specific implementation of the pattern to evaluate the entire pattern. Because specific implementations are unusable, the relevant pattern attributes can not be directly measured in a quantitative way.

Pattern evaluation adds retrospect and the knowledge of many experts to existing pattern documentation. This study also relates to software architecture as it solves a problem found in the software pattern selection process. Software pattern evaluation helps when performing pattern oriented software architecture in cases where alternative patterns to solve the same problem and only a single pattern can be selected. This is an important factor to take into account, because it means that rather than selecting individual patterns, an architect will want to select an architectural style, and thus select a large set of patterns that fit this style. This area of software architecture has developed, which resulted in a large amount of documented patterns and allows for comparing architectural styles [5].

Although it seems that comparing individual patterns is less relevant for software architecture, an architectural style is selected at the early stages of software design and cannot easily be changed after the development has started. This creates a problem because while the software is being developed, the requirements for the project or the environment will change. Therefore it is necessary to extend the architecture or at times alter the existing architecture. At this point it becomes relevant to compare individual patterns in order to select the pattern that fits the project requirements. This is an ongoing process that happens throughout software development and relies on the experience of software architects and developers. Current documentation of software patterns is lacking a way to compare them with each other. But, if multiple patterns tackle the same problem, how does an architect decide which one to use? This is tacit knowledge of experienced architects and developers, leading to the following problem statement: *"There is no formal way to express the quality of one pattern over another"*.

This paper presents the Software Pattern Evaluation Method (SPEM). Using SPEM, software producing companies are supported in pattern selection decision making and are able to quantitatively compare different patterns. SPEM enables them to get an overview of specific pattern characteristics in a timely manner. Also, SPEM can be used to generate a publicly available pattern related body of knowledge, helping research and practitioners in architectural research and decision making. This paper first gives an overview of research related to pattern comparison in Section II. The design science approach used in the research is described in Section III, after which SPEM is presented in Section IV. The pattern evolution, including the initial method creation (Section V-A) and method evaluation (Section V-B), showing the changes during the method

 ISBN: 978-1-61208-343-8

creation process can be found in Section V. To conclude, the applications of SPEM are discussed (Section VI), followed by a conclusion (Section VII).

II. Related Work

Software Patterns — As software development was maturing in the 1980s, the need arose to share common solutions to recurring problems. This process started out by developers communicating to their colleagues how they solved a recurring development issue. The communication was informal and there were no clear rules for documentation. In later years, software patterns have become an essential part of software development as a way to capture and communicate knowledge. Software patterns are solutions to a recurring problem in a particular context [6], [7]. When properly documented these solutions are a valuable asset for communication with and among practitioners [8]. Usage of software patterns allows for time and cost reduction in software development projects, making them an important tool for software design and development. Although software patterns started out as a way to communicate solutions among developers, they have become a crucial part of software architecture [9] as well. A pattern selected by a developer, however, does not take into account the entire architecture and how it combines with existing patterns. This problem is solved by selecting patterns at the architectural level.

Architecture Evaluation — Evaluation is commonly used in software architecture in order to increase quality and decrease cost [10]. Many evaluation methods for software architecture have been developed and compared in recent years [11]. The evaluation should be performed as early as possible in order to prevent large scale changes in later stages of development. Software architecture evaluation is linked to the development requirements and desired quality attributes. Therefore, it is not a general evaluation of software architecture, nor an evaluation of a specific implementation. The evaluation should be an indication of whether the proposed architecture is a good fit for the project. Pattern comparison and evaluation has been done before [12] in a quantitative manner, but has focussed on the *implementation* of different patterns and lacks the evaluation of the *idea* the pattern describes.

III. Research Approach

This section presents the research questions answered in this paper and the design science approach used to construct SPEM. The main research question (MRQ) answered in this paper is:

> ***MRQ:*** *How can software patterns be evaluated in a manner that is objective and allows for comparison?*

The aim of this study is to aid software architects in the decision making process of selecting software patterns. This can only be useful when the evaluation method yields objective results. Since a software pattern can not be objectively measured in any way, the opinions of multiple software architects are used in the form of scores. A quantitative study also allows for easy comparison between alternative patterns. For the purpose of answering this research question, multiple sub-questions are constructed:

> ***SQ1:*** *Which attributes are relevant in pattern evaluation?*

Rationale: Patterns can possess many attributes that give important information on usefulness and quality. For example, how the pattern effects performance or maintainability can both be attributes of a pattern.

Attributes are used in software architecture to evaluate the quality of certain aspects of the architecture. We apply the same principles for evaluation of software patterns. The first step is to create a list of attributes by looking at related literature. This list is then reduced by performing expert interviews. This tells us which of the listed attributes are important to software architects when evaluating a pattern. A validation of the reduced list of attributes is performed by interviewing a second expert. The result of these interviews is a validated list of attributes relevant in the software pattern evaluation process.

> ***SQ2:*** *How can attributes relevant in pattern evaluation be quantified in a manner that allows for comparison?*

Rationale: Typical documentation on software patterns is qualitative in nature. Although this might be suited for documentation on patterns it does not allow for comparison. For this reason, the different attributes relevant for pattern evaluation need to be quantified. A structured method of quantification that is used for evaluations would allow for patterns to be compared on attribute level.

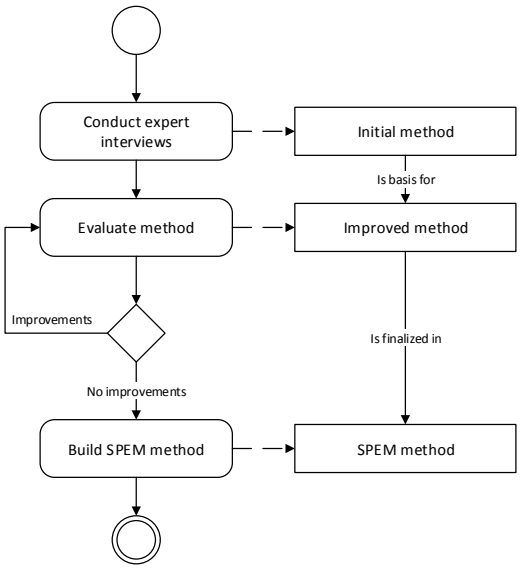


Fig. 1. Design Science Research Method

To answer this question we first look at comparable methods of quantification within the domain of software engineering. From these methods the specific characteristics are deduced. An example of these characteristics can be the

ability to assign a negative value to an attribute. A method for quantification is constructed based on the list of characteristics. The method is evaluated by using it in a focus group session after which is can be incrementally improved.

A design science approach is used, which is depicted in Figure 1. An initial method is created based on an earlier exploratory study [13], extended by expert interviews. The method is evaluated in multiple cycles in which the method was put to practice in a real-world setting. Three subsequent sessions are organized in which both professional software architects and software architecture students used the method. Information system master students can be used as test subjects instead of professional software developers [14]. All sessions were recorded and an evaluation form is filled in by all participants after each session. A revised method was constructed after each session, based on the feedback, which is used as input for the next session. After three sessions no significant changes were needed any more, leading to the creation of the final method (i.e. SPEM).

IV. SPEM - Software Pattern Evaluation Method

SPEM has been constructed to evaluate software patterns in a manner which allows for comparison. There are two distinct roles:

Evaluator — Leads the evaluation process by introducing concepts and directing discussions. Is responsible for timekeeping, collecting all deliverables and noting scores.

Participant — A software architect or developer who uses his knowledge to assign scores to attributes, enters discussion, shares arguments and tries to reach consensus.

The evaluation data is gathered during a focus group session. These sessions vary in duration from one to two hours. Four to twelve participants can partake in the evaluation, excluding the evaluator. The basis of the evaluation are attributes, categorized in both quality attributes and pattern attributes. Quality attributes are used to measure the impact the pattern has on software quality and are based on ISO/IEC 25010 [15]. The following quality attributes (excluding sub-attributes) are used in SPEM:

- **Performance efficiency —** Degree to which the software product provides appropriate performance, relative to the amount of resources used, under stated conditions.
- **Compatibility —** The ability of multiple software components to exchange information or to perform their required functions while sharing the same environment.
- **Usability —** Degree to which the software product can be understood, learned, used and attractive to the user, when used under specified conditions.
- **Reliability —** Degree to which the software product can maintain a specified level of performance when used under specified conditions.
- **Security —** The protection of system items from accidental or malicious access, use, modification, destruction, or disclosure.
- **Maintainability —** Degree to which the software product can be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
- **Portability —** Degree to which the software product can be transferred from one environment to another.

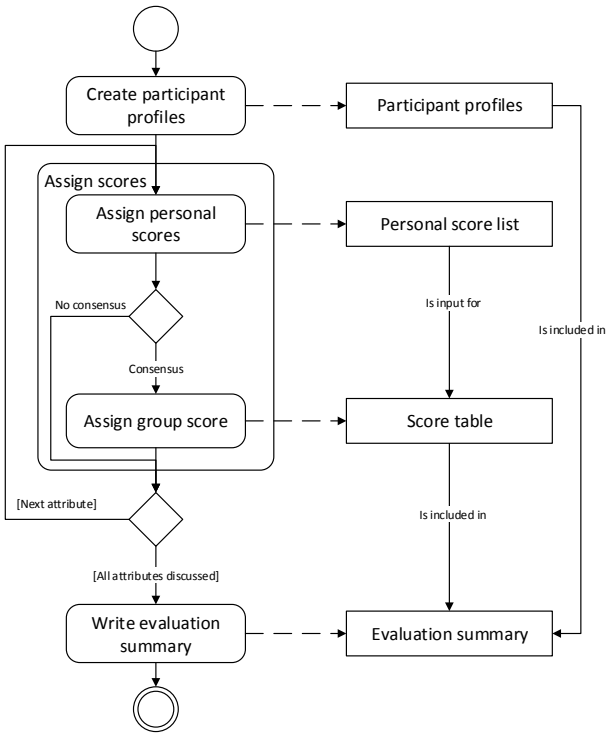
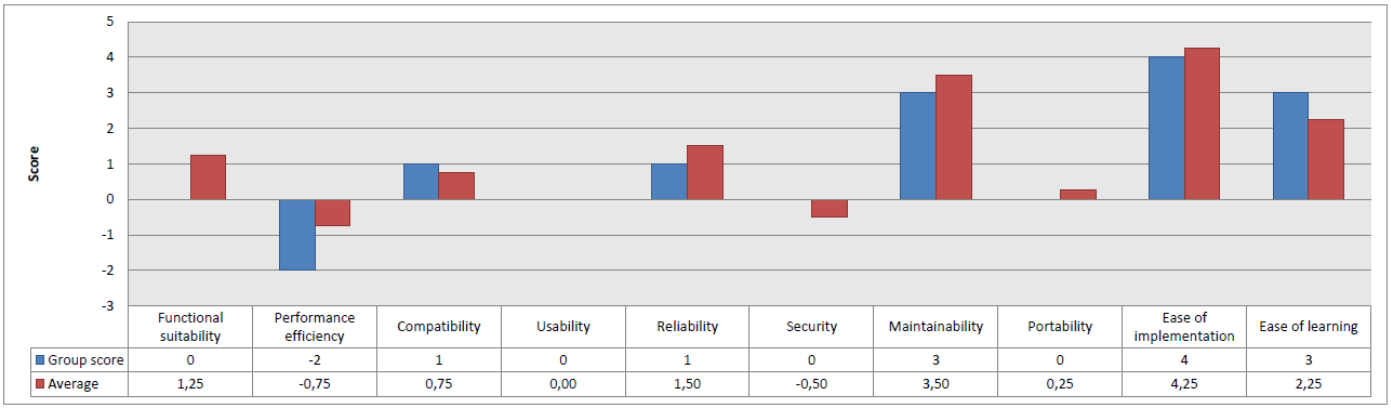


Fig. 2. SPEM: Software Pattern Evaluation Method

Pattern attributes are characteristics of the pattern itself, used to measure its learnability or ease of implementation. The goal of the evaluation is to assign a score to each attribute by all participants. The score is a relative measure based on the experience of the participant, ranging from -3 to $+3$. The score is a generalization of the software pattern, not based on a specific implementation. Experience using the pattern in a variety of situations is expressed by the score. Therefore the difference in experience among all participants is a key factor in the evaluation, which is compensated in a group score. A group score is assigned to each attribute (excluding sub-attributes) and expresses a score after a round of discussion. The discussion of each attribute allows the participants to share their knowledge with each other. The goal of the discussion is to reach consensus, meaning that after knowledge has been shared between participants with different amounts of experience, one score is assigned on which all participants agree. The result is quantitative data in the form of scores based on personal experience and the knowledge of a group, visualized in an evaluation summary (see Figure 3).

SPEM consists of four activities and four deliverables, as shown in Figure 2. The first activity focuses on creating participant profiles [16]. These profiles are forms containing fields for the participant's name, job description and years of experience. Additionally, there are input fields for the pattern name and experience with the pattern. Provided with the participant profile is a personal score list containing a list of quality attributes, sub-attributes and pattern attributes. For

 ISBN: 978-1-61208-343-8



	Functional suitability	Performance efficiency	Compatibility	Usability	Reliability	Security	Maintainability	Portability	Ease of implementation	Ease of learning
Group score	0	-2	1	0	1	0	3	0	4	3
Average	1,25	-0,75	0,75	0,00	1,50	-0,50	3,50	0,25	4,25	2,25

Fig. 3. SPEM evaluation summary (observer pattern)

each item on this list there is a possibility to give a personal score. The evaluator introduces the method to the participants by explaining each deliverable and the focus group session protocol. In the protocol, all activities and by who they are performed are listed and described. Thereafter, the evaluator asks the participants to fill out the participant profile.

In the second process, personal scores are assigned to an attribute. During the evaluation the scores are recorded in the personal score list. After the evaluation the personal scores are entered in the score table. The score table contains rows with all attributes used in the evaluation and columns containing all personal scores, average scores, standard deviations and group scores. The evaluator introduces an attribute by giving a short description. The participants are then asked to assign a score to the attribute and all corresponding sub-attributes.

In the next phase, a group score is assigned to an attribute and noted in the score table. The group score is a score which is produced by gaining consensus. This means all participants partake in a discussion. The focus of the discussion is to exchange arguments on the score of an attribute. If consensus is reached among all participants, the resulting group score is assigned and noted on the score table. If consensus is not reached, the group score is not assigned and no score will be noted in the score table. The evaluator initiates a discussion on the current attribute by asking a single participant's score and motivation for the score. Other participants are free to respond and exchange views, directed by the evaluator. If the discussion ends or if no time is left, the evaluator asks the participants if they have reached consensus. When consensus is reached, the group score is recorded in the score table.

When all attributes have been evaluated, an evaluation summary is created. The evaluation summary is a combination of all participant profiles and a filled out score table. Additionally a new form is added containing the name of the evaluator, date and threats to validity. This gives the evaluator the opportunity to note any occurrences that are not expressed in the main deliverables. This process is performed by the evaluator at the end of the focus group session and concludes the evaluation.

V. Method Evolution

This section discusses how the initial method evolved and shows the explicit changes made to the method based on the expert evaluation sessions.

A. Initial Method Construction

Expert interviews formed the basis of the initial version of the SPEM method. Two software architects from different companies cooperated to share their views on software pattern evaluation. Understanding which attributes are relevant in pattern evaluation and how they could be quantified was the goal of the interview. During the interview a list of quality attributes derived from ISO/IEC 9126 [17] and ISO/IEC 25010 [15] was discussed, the latter being preferred by the interviewees. Although both interviews had different results on the importance of each individual attribute of the standard, none could be excluded. Ease of learning and ease of implementation are both attributes describing characteristics of software patterns. Both these attributes should be included in software pattern evaluation as they play an important part in software pattern selection.

Scenarios are often used in software architecture evaluation, but do not fit pattern evaluation. The fact that patterns are evaluated without a specific implementation in mind makes the use of scenarios irrelevant. A software architect should interpret the results of pattern evaluation by relating it to their own project. When attributes are quantified using a score, it should be possible to assign a negative value. Patterns can affect software quality in a negative way or have negative characteristics, which a score should be able to express. The range of the scores should be between a five and ten point scale. At larger ranges it would be difficult for an architect to assign an accurate score.

When multiple architects perform a pattern evaluation, they are likely to have varying degrees of experience. Experience is key in understanding software patterns and their effect on software quality. It is important to assign a score to an attribute which takes into account the varying degrees of experience software architects have. This should be done using discussion and consensus. In a discussion, those who have more experience can share their knowledge with those who have less experience. Together working towards consensus can improve the level of knowledge of the participants and consequently improve the score. Software pattern evaluation should be performed with at least one architect who has

 ISBN: 978-1-61208-343-8

experience using the pattern which is being evaluated. This restriction makes sure the evaluation yields a valuable result.

Based on these interview results a method was constructed incorporating the following:

- All attributes and sub-attributes from ISO/IEC 25010 [15].
- Two additional attributes; *ease of implementation* and *ease of learning*.
- Scoring ranging from -5 to $+5$.
- Discussion after each attribute.
- The goal of trying to reach consensus on each attribute.

B. Method Evaluation

Using a design science approach the initial method was evaluated and improved several iterations. A total of three focus group sessions were hosted to evaluate the method. In these sessions the method was carried out by evaluating a software pattern. Participants were asked to fill out an evaluation form at the end of the focus group session. The feedback gathered in the evaluation forms and experiences from hosting the sessions were the basis for each new iteration of the SPEM method.

During the first focus group session, four software architects participated, each having over nine years of experience in software development. During this session the observer pattern was evaluated using the initial version of SPEM. The first attribute, functional suitability and corresponding sub-attributes raised many questions. It was not possible to assign a score, as the attribute demanded a specific context. Not having a description for sub-attributes was confusing and diverted discussions to the definitions of certain sub-attributes. Based on the evaluation session, the following improvements were incorporated in the method:

- **Removal of attribute 'Functional suitability' —** This attribute, including its sub-attributes turned out to be irrelevant based on the focus group session. Functional suitability can only be assessed by looking at specific implementations.
- **Including a description for all sub-attributes —** A description of each attribute, including all sub-attributes was needed. This way different interpretations of attributes can be precluded.

The second focus group session was performed with twelve participating master students. The students have an information systems background and were all enrolled in the Software Architecture course, which prepared the students for the focus group session. The Access Point pattern was evaluated and participants were free to discuss attributes without any intervention from the evaluator. This resulted in lengthy discussions making the session take longer than anticipated. Discussions should be halted by the evaluator after a certain period of time based on the time that is available.

Assigning scores to sub-attributes and discussing them was time consuming. Sub-attributes needed a less prominent role in the method. It was not always possible for participants to assign a score to an attribute. Therefore it should be possible to have an explicit option stating that no score is assigned, instead of leaving it empty which might imply a neutral score. The introduction of the pattern was unclear, leading to discussion and debate. How scores should be assigned and what they represent raised questions during the session. The evaluator should focus more on explaining the meaning of scores and the difference between quality attributes and pattern attributes.

Based on the second evaluation session, the following improvements were incorporated in the method:

- **Sub-attributes removed —** Because discussion on sub-attributes took too long, they were removed from the method.
- **Added an option to give an attribute no score —** An explicit way was added for participants to indicate they do not want to give a score to a certain attribute.
- **More focus on pattern introduction —** The pattern needs to be thoroughly explained to prevent discussions.
- **More focus on explaining what the scores represent —** Scores represent the impact the evaluated pattern has on software quality or characteristics of the pattern itself. This distinction needs to be clear in order to properly assign scores.
- **Stronger role of the evaluator —** The evaluator needs to direct the discussions. Apart from initiating discussions, they should also be halted. Time keeping is the responsibility of the evaluator.

The third focus group session was performed with different students from the same group as the second focus group session. Although sub-attributes did not receive a score, they were referred to in discussions to better understand an attribute. Therefore it is important to include the sub-attributes in the method. Discussions for each sub-attribute would increase the time to complete an evaluation substantially. A personal score should be assigned to a sub-attribute while discussions and consequently group scores, should be left out.

Based on the third evaluation session, the following improvements were incorporated in the method:

- **Sub-attributes added —** Can help the understanding of attributes and provide more detail to the data.
- **Sub-attribute discussions removed —** Gives the sub-attributes a less prominent role in the method and focusses more on attributes.
- **Descriptions for each attribute / sub-attribute added to participant profile —** Allows the participants to read descriptions of attributes independent of the evaluator.

After the three sessions, the final method (i.e. SPEM) was created.

VI. SPEM Implementation

SPEM is created to evaluate software patterns in general, without a specific implementation in mind. This enables the option for comparison of software patterns, because each pattern has been evaluated as an abstract solution. It prevents unbalanced comparison between patterns based on different implementations. There is a trade-off between easy to compare generic evaluation and implementation specific evaluation. An implementation specific evaluation provides more accurate

 ISBN: 978-1-61208-343-8

data, but it can only be compared to evaluated patterns based on the same implementation. A generic evaluation might not be as accurate, but ensures all evaluated patterns can be compared. SPEM can be used for implementation specific evaluation with few adjustments. It requires the evaluator to explain that the scores should be assigned with an implementation in mind. There needs to be an input field describing the implementation on the score table. With these adjustments an evaluation session would be identical to SPEM and allows for use of all processes and deliverables used in SPEM.

This study provides knowledge on software pattern evaluation by introducing a method to evaluate software patterns. The data SPEM evaluations provide further expands the body of knowledge on patterns. It adds retrospect to the existing software pattern documentation and provides insight on the impact patterns have on software quality. A collection of SPEM evaluation results provides valuable knowledge on the understanding of software patterns and software quality. A knowledge base would enable the disclosure of SPEM evaluation results and would allow results to be combined and compared. From an industrial perspective, a SPEM knowledge base would enable software architects to share their knowledge on software patterns. It would make knowledge available to aid in software pattern selection, leading to better decision making and overall software quality. It is through sharing knowledge that software pattern selection can reach a higher level of maturity, allowing for a structured way of comparing software patterns.

SPEM uses discussion and consensus to obtain quantitative evaluation data. This method of quantification was introduced to cope with different experience levels among participants. It has imposed a constraint on the method of data gathering used in SPEM. As discussions require interaction between participants, all participants need to be able to communicate with each other at the same time. Therefore SPEM is used in focus group sessions, limiting the number of participants. A trade-off exists between a more accurate score based on consensus with a small number of participants and a less accurate, but more reliable score with a large number of participants.

VII. Conclusion

SPEM is an objective software pattern evaluation method which can be used to compare patterns. It is used to evaluate relevant attributes of patterns based on the experience of software architects. SPEM provides quantitative data on attributes in the form of scores. The data can be interpreted and visualized to allow for software pattern comparison. This answers the main research question (MRQ).

The question "Which attributes are relevant in pattern evaluation?" (SQ1) is answered with a list of attributes, consisting of quality attributes and pattern attributes. The quality attributes are based on ISO/IEC 25010 and modified for pattern evaluation, resulting in the following set of attributes: *a*) Performance efficiency, *b*) Compatibility, *c*) Usability, *d*) Reliability, *e*) Security, *f*) Maintainability, and *g*) Portability. These attributes can be quantified in a manner that allows for comparison (SQ2) by rating the different attributes by experts in a focus group setting. It requires that personal scores ranging from -3 to +3 are assigned to all attributes and sub-attributes. A group score is assigned to all attributes after a discussion and reaching consensus. All scores are noted in the score table. A future step is to perform SPEM sessions to gather data and produce results allowing for software pattern comparisons. Gathering the output of evaluation sessions and adding them to the knowledge base can help to further validate the method and produces valuable knowledge for both academic and industrial purposes.

Acknowledgment

The authors would like to thank Leo van Houwelingen from Exact Software and Raimond Brookman from Info Support for their valuable input. We also want to thank the architect team from Info Support for their cooperation. This research is funded by the NWO 'Product-as-a-Service' project.

References

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice, 2/E*. Pearson Education India, 1998.

[2] A. Jansen, J. Van Der Ven, P. Avgeriou, and D. K. Hammer, "Tool support for architectural decisions," in *The Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. IEEE, 2007, pp. 4–4.

[3] M. A. Babar and I. Gorton, "A tool for managing software architecture knowledge," in *Proceedings of ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*. IEEE, 2007, pp. 11–17.

[4] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *Software, IEEE*, vol. 22, no. 2, pp. 19–27, 2005.

[5] G. Booch, "On creating a handbook of software architecture," in *Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, vol. 16, no. 20, 2005, pp. 8–8.

[6] F. Buschmann, *Pattern oriented software architecture: a system of patterns*. Ashish Raut, 1999.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[8] K. Beck, R. Crocker, G. Meszaros, J. Vlissides, J. O. Coplien, L. Dominick, and F. Paulisch, "Industrial experience with design patterns," in *Proceedings of the 18th international conference on Software engineering*. IEEE Computer Society, 1996, pp. 103–114.

[9] F. Buschmann, K. Henney, and D. C. Schmidt, "Past, present, and future trends in software patterns," *IEEE Software*, vol. 24, no. 4, pp. 31–37, 2007.

[10] G. Abowd, L. Bass, P. Clements, R. Kazman, and L. Northrop, "Recommended best industrial practice for software architecture evaluation," DTIC Document, Tech. Rep., 1997.

[11] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Proc. of the Australian Software Engineering Conference*. IEEE, 2004, pp. 309–318.

[12] M. Hills, P. Klint, T. Van Der Storm, and J. Vinju, "A case of visitor versus interpreter pattern," in *Objects, Models, Components, Patterns*. Springer, 2011, pp. 228–243.

[13] J. Kabbedijk, M. Galster, and S. Jansen, "Focus group report: Evaluating the consequences of applying architectural patterns," in *Proc. of the European conference on Pattern Languages of Programs (EuroPLoP)*, 2012.

[14] M. Höst, B. Regnell, and C. Wohlin, "Using students as subjectsa comparative study of students and professionals in lead-time impact assessment," *Emp. Softw. Engineering*, vol. 5, no. 3, pp. 201–214, 2000.

[15] ISO/IEC, *ISO/IEC 25010. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, 2010.

[16] R. Donselaar and J. Kabbedijk, [Accessed: 2014-04-08]. [Online]. Available: http://www.staff.science.uu.nl/ kabbe101/PATTERNS2014

[17] ISO/IEC, *ISO/IEC 9126. Software engineering – Product quality*, 2001.

 ISBN: 978-1-61208-343-8