Comparing software patterns

SPEM: A Software Pattern Evaluation Method

René van Donselaar
Student ID: 3780570

Utrecht University, Utrecht, The Netherlands
r.n.a.vandonselaar@students.uu.nl

Content

1. Problem statement

Modern software architecture relies heavily on the use of many software patterns. These patterns often extend each other or work together to solve one problem. Pattern languages are groups of patterns that together target a style of architecture (Buschmann, Henney, & Schmidt, 2007). 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 (Booch, 2005; Shaw & Garlan, 1996). Although it seems that comparing individual patterns is less relevant for software architecture, an architectural style is selected at the early stages of software architecture 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. We have formulated the following problem statement:

Software patterns and their impact on software quality cannot be compared without studying pattern documentation or relying on relevant experience.

Current documentation of software patterns is lacking a way to compare them with each other, while their influence on software design and on software quality is strong (Brito e Abreu & Melo, 1996). But if multiple patterns tackle the same problem, how does an architect decide which one to use? At this time such decision is based on the experience of architects and developers (Garlan, 2000).

There are a few reasons why pattern documentation is not well suited for comparison:

- **Different description styles**

Patterns are often written by different authors, who can impose a different style of writing on documentation. For a comparison, this means that the reader cannot know what content documentation will hold. One author might give an example about how the pattern can be implemented in a certain project, while another author does not. Aside from the formal notation, the author has freedom in the details and examples found in pattern documentation. A comparison requires that aspects which are important to the reader are consistently mentioned in all alternatives he tries to explore. Although the majority of patterns are written by different authors, there are some exceptions to this.

Pattern languages are a collection of patterns which are written by an author to coherently solve a problem using multiple patterns. However, when we try to compare

patterns we are often looking at alternatives, not patterns that are part of a pattern sequence. For example, it is unlikely one would compare the role pattern with the limited view pattern (Yoder & Barcalow, 1998). Both patterns can work together in sequence, but solve a different problem. The role pattern allows for users to be assigned a role, which can be used to restrict the user in their access of the application's functions, while the limited view pattern uses that the role information to either show or not show options to the user in the user interface. Both patterns are no alternative to one another as they solve a different problem, which means that their comparison is not beneficial to architectural decision making.

Another exception is a *pattern collection*, which is written by the same author or group of authors. These collections can be found in software architecture handbooks and academic literature. There are also examples of (large) companies that mine their own patterns in order to allow for design reuse within their own company (Beck, et al., 1996). Pattern collections have the advantage that they are written by the same author(s) and as opposed to pattern languages, can also include alternative patterns. This means that pattern collections can provide a way to compare alternative patterns based on documentation. However, the added restriction is that we are limited to the patterns included in the collection.

- **Qualitative description**

Pattern documentation is qualitative in nature, consisting of text, images and diagrams. Text is written using a formal notation, dictating the number of sections and their type content. Pattern documentation usually tries to answer a few questions:

- What is the problem?
- Why is it a problem?
- What is the solution?
- What are the consequences?

At minimum the chosen notation answers the above mentioned questions, but might require more sections. Additional sections are often used to give examples of the pattern and to explain how the pattern can be used for a certain type of implementation. Images can be used to support the text. For instance, a real world example of the façade pattern can be supported by an image of a support desk employee who allows the user to access complex services without ever knowing these services exist or how they work, in this case the image would help the reader to understand the pattern. Patterns often describe objects are their relationships, supported by class diagrams. Any quantitative data on the pattern is absent, which might be explained by the fact that a pattern is an abstract solution without details on implementation. Therefore any quantitative data on the patterns consequences would be irrelevant, because it would be implementation specific. A quantitative expression of a pattern's characteristics or impact on software quality would have the reader ask, 'in which case would these numbers apply?' and 'what do these figures mean for my implementation?'. However, the goal of pattern documentation is to communicate working design, not to compare. The qualitative nature of pattern documentation makes it difficult to compare patterns. It means that the documentation has to be studied in-depth, before it can be compared.

- **Quality attribute description**

Software patterns can have an impact on software quality, which can be divided in multiple quality attributes. Quality attributes can be used in software architecture evaluation and are well a well-known concept to software architects. There are many different lists mentioning a number of quality attributes and sub-attributes. Examples of these attributes are performance, maintainability and usability. Quality attributes are often mentioned in pattern documentation, specifically in the forces and consequences sections. However, the level of depth in which these quality attributes are mentioned is up to the author. It can be the case that performance is mentioned extensively, while other attributes like maintainability are briefly mentioned, or not mentioned at all. Therefore the reader of pattern documentation cannot know what to expect, meaning that the documentation has to be studied. Time is lost if the pattern documentation does not contain quality attributes that interest the reader. Pattern documentation does not give a clear overview of the impact that a pattern has on software quality and its tradeoffs. The fact that the author has freedom in which quality attributes are mentioned, means that pattern documentation is not a suitable tool for software pattern comparison based on quality attributes.

In the remainder of this section the need to compare patterns in explained. We have discerned four scenarios that occur in software pattern selection (Figure 1). In the first scenario the architect wants to select a pattern that solves a particular problem and only one pattern provides a solution. In this scenario no choice is presented and it is clear which pattern should be selected. In the second scenario the architect is aware of two patterns that solve the problem. Because he has experience using both patterns, he uses his existing knowledge of previous implementations to make a well-founded decision. In the third scenario multiple patterns solve the problem. The architect has experience using one solution, while he has no experience with the other. This makes for an interesting decision, as one would expect an architect would study the pattern he has no experience with in order to make a decision. However, it often occurs that an architect will select a pattern he is familiar with in favor of an unused alternative. Studying a pattern requires an investment from the architect, which makes a known pattern a more appealing solution. This is a problem, because a known solution is not always the best solution. The architect does not have experience using either of the patterns presented in the fourth scenario. In this case both patterns have to be studied in-depth in order to make a decision, making it time consuming.

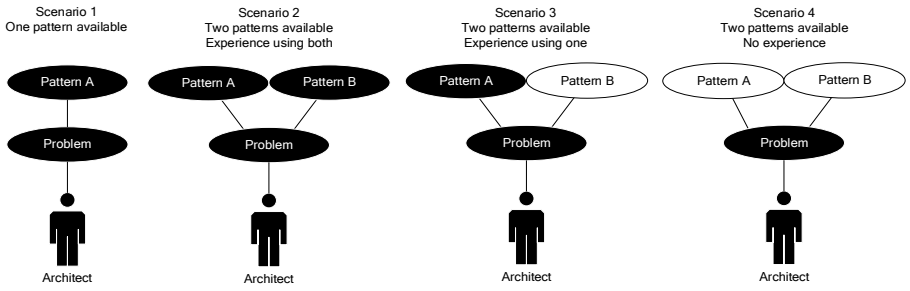


Figure 1 - Scenarios in software pattern selection

Only the second scenario involves a decision based on experience. However, there are too many pattern available for an architect to have experience with. For inexperienced architects the third and fourth scenario are common. In this study we want to capture the knowledge of those who have experience with a pattern in order to share it with those who do not.

2. 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 (Bass, Clements, & Kazman, 1998). 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 (Jansen, Van Der Ven, Avgeriou, & Hammer, 2007). 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 (Babar & Gorton, 2007). 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 be to created enabling the evaluation and documentation of crucial attributes of a software pattern (Tyree & Akerman, 2005). 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 cannot 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 (Booch, 2005).

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 thesis 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.

2.1 Thesis structure

This thesis first explains why software pattern evaluation is needed and what makes the lack thereof a problem in section *1 Problem statement*. In section *2 Introduction* an introduction to software patterns is given and definitions for key concepts are explained. The research questions are introduced and explained in section *3 Research approach*. The design science approach used in this research is depicted and explained, after which the section is concluded with an explanation of validity assurance and an interview protocol. Section *4 Literature review* gives an overview of research related to pattern evaluation and comparison. An initial version of the pattern evaluation method is presented and discussed in section *5 Initial method construction*. The section is divided in three subsections, covering two interviews and their implications for software pattern evaluation. In section *6 Method evaluation* the evolution of the method is described by reporting on three focus group sessions, each covered in a sub-section. The final version of the pattern evaluation method (SPEM) is presented in section *7 SPEM – Software Pattern Evaluation Method*. In section *8 SPEM implementation* we discuss the various implementations of SPEM and its use in academia and the software industry. The restrictions of the method, the advantages and how SPEM evaluations should be interpreted is discussed in section *9 Discussion*. The thesis is concluded with section *10 Conclusion and future work*, in which the research questions are answered and future work is discussed.

2.2 Definitions

2.2.1 Software pattern

A software pattern is a solution to a recurring problem in a particular context (Schmidt, 1995; Gamma, Helm, Johnson, & Vlissides, Gang of four, 1995; Buschmann, 1999). When properly documented these solutions can be shared with the industry. 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 (Buschmann, Henney, & Schmidt, 2007). Patterns were popularized by the "Gang of Four" book in which a collection of patterns was documented along with a formal notation (Gamma, Helm, Johnson, & Vlissides, 1995). Since then many patterns were documented. An example

of this is the work of Grady Booch, who documented thousands of patterns and architectural styles (Booch, 2005). Currently it is nearly impossible to design software without the use of software patterns.

2.2.2 Pattern language

A set of patterns which are related and have a common goal are referred to as a pattern language (Roberts, 1996; Meszaros, 1998; Alexander, 1977). A pattern language can help the software architect in the pattern selection process. Instead of having to select individual patterns, a pattern language provides a collection of patterns and documentation on their implementation respective to each other. When a pattern requires another pattern to function, it is important to know in which order they should be implemented. In a pattern language not only the pattern itself is described but also which patterns might have to be implemented before and after. The order in which the patterns need to be implemented is called a pattern sequence (Buschmann, Henney, & Schimdt, 2007). For example, the limited view pattern only shows functionality the current user is allowed to access which requires the session and role pattern to function, as it would provide data on who is using the system. In sequence the session pattern would be implemented first, followed by the role and limited view pattern.

When selecting individual patterns there might be overlap, as each pattern can be documented by a different author. This causes the selection of individual patterns to be time consuming because each pattern has to be studied in-depth to see if it can be implemented with other patterns. In a pattern language there is no overlap between patterns, they were documented to be part of a whole and the work of a single author. Each pattern is essentially part of a puzzle, all connected together in a certain way and are cut out to fit exactly.

The characteristics of pattern languages make them valuable for software architects when selecting patterns at the initial stages of development. However, when the architecture is adapted it becomes less likely a pattern language can be used in its entirety. When the architecture is adapted or expanded, the existing architecture has to be taken into account. Although the patterns in a pattern language work well together, it does not take into account other patterns or pattern languages. Therefore the same problem arises as with the selection of individual patterns. There can be overlap or certain patterns cannot be combined. It means that a pattern language has to be studied in-depth when implementing together with other patterns or pattern languages.

2.2.3 Architectural style

An architectural style limits the architect in his choices for architectural elements and can be defined as a constraint to both the design elements and the formal relationships among design elements (Perry & Wolf, 1992). This means that, just like style in general, architectural style restricts which elements can be used in order to create coherency. In software architecture an architectural style is used to make a system have certain properties and satisfy non-functional requirements.

An architectural style can also be referred to as an architectural pattern (Bass, Clements, & Kazman, 1998; Buschmann, 1999). An example of such a pattern, which can impose a style on software architecture, is REST (Khare & Taylor, 2004). However, there are also definitions of architectural style that differentiate it from an architectural pattern. An architectural style can be viewed as being more predominant than an architectural pattern (Bosch & Molin, 1999). Within the software engineering community there is no consensus on the definition of architectural style as discussed in this section. In this thesis we consider an architectural style and architectural pattern to be synonymous as defined by (Bass, Clements, & Kazman, 1998) and (Buschmann, 1999).

3. Research approach

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

MRQ: How can software patterns be evaluated by software architects 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 cannot 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 for software 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. Quality 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. If validation fails, another expert interview and validation is performed. When successful, the result of these interviews is a validated list of attributes that relevant in the software pattern evaluation process.

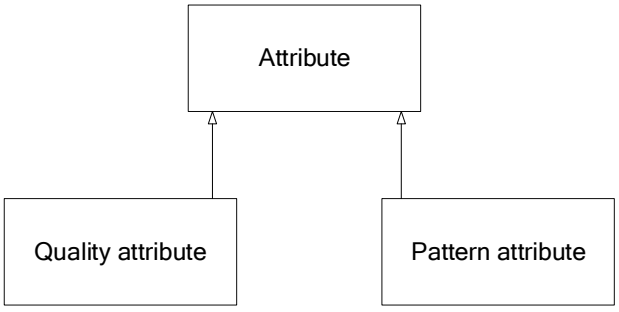


Figure 2 - Attribute, Quality attribute and Pattern attribute

An attribute can refer to a quality attribute or a pattern attribute. This distinction is made because a quality attribute is not a property of a pattern, but rather an aspect of software quality. When a quality attribute is referred to in the context of pattern evaluation, we are talking about the impact the pattern has on software quality. A pattern

attribute refers to a characteristic of the pattern itself, for example the ease at which the pattern can be learned. Learnability as a quality attribute would express to what degree the pattern influences the software in this aspect, while it does not say anything about the learnability of the pattern itself. Although quality attributes and pattern attributes are viewed from a different perspective, they have in common that they are attributes of a pattern which can be relevant for evaluation. Therefore the term attribute is used, a characteristic of a pattern which can be relevant for pattern evaluation. The answer to SQ1 would be a list of attributes which are relevant for pattern evaluation. It can be any number of quality attributes are pattern attributes or mixture of both.

SQ2: How can attributes relevant for 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.

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. Finding out which characteristics are important to architects when evaluating a pattern is the next step. This is done by conducting an expert interview. In this interview the software architect can express which characteristics are important and why. A second interview is held with a different expert to validate the findings. The result is a validated list of characteristics that are important for quantification of attributes. 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.

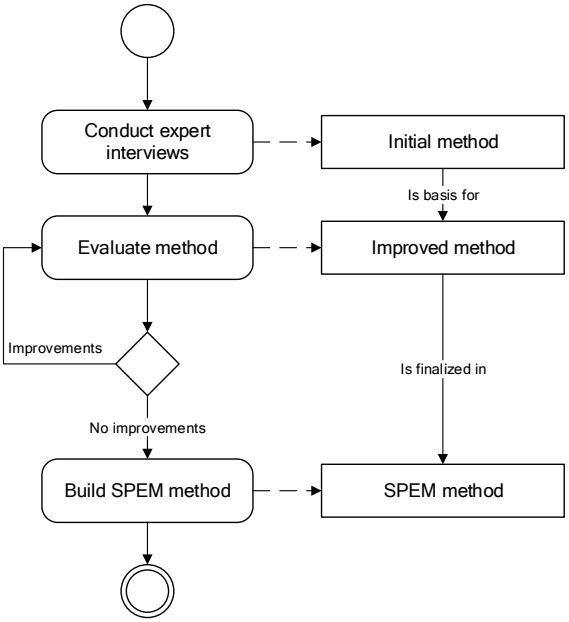


Figure 3 - Design Science Research Method

Table 1 - Design Science Research Method Activity Table

Activity	Description
Conduct expert interviews	Interviews with experienced software architects are conducted to form the basis of the INITIAL METHOD.
Evaluate method	The method evaluated by using it in a focus group session, gathering the feedback with evaluation forms. This process results in an IMPROVED METHOD.
Build SPEM method	Based on the IMPROVED METHOD, the finalized SPEM METHOD is built.

Table 2 - Design Science Research Method Concept Table

Concept	Description
Initial method	Initial version of the method created based on expert interviews.
Improved method	Improved version of the method based on focus group session feedback.
SPEM method	Software Pattern Evaluation Method.

A design science approach is used, which is depicted in Figure 3. An initial method is created based on an earlier exploratory study (Kabbedijk, Galster, & Jansen, 2012), extended by expert interviews. The interviews are based on the knowledge gained from literature and are aimed towards answering SQ1 and SQ2. After conducting the first interview an analysis follows and a second interview is scheduled. The questions from the second interview are used to validate the answers of the first interview, but are also an opportunity to ask questions that arose based on the first interview. The result of the interviews is a list of relevant attributes for pattern evaluation and a way to quantify them. This information is used to construct a first version of the evaluation method which we call M_1. Thereafter the method is evaluated in multiple focus group sessions in which the method is put to practice in a real-life setting (Figure 4).

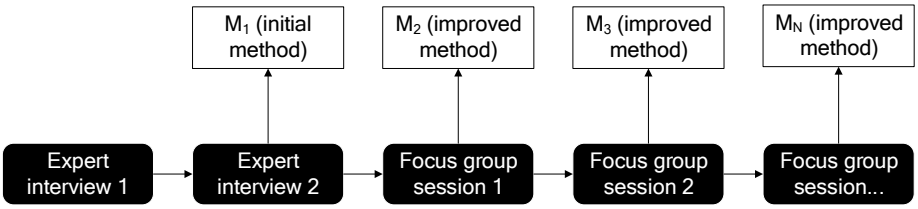


Figure 4 - Method construction process

These sessions will be performed with software architects or software architecture students. As Höst points out that information system master students can be used as test subjects instead of professional software developers (Höst, Regnell, & Wohlin, 2000). In the first evaluation session the initial method (M_1) is executed and evaluated. The session is recorded and an evaluation form is filled out by all participants after the session. Improvements to the method are added and a new improved method (method increment) is constructed (M_2). A new version of the method is always constructed, regardless of the number of changes made. Even if no changes are made, it is still important to capture the rationale and explain why certain feedback did not lead to change. When changes are made to the method, it is depicted in a diagram, showing where the new version differs from the old version. The diagram is followed with an in-depth analysis and textual description of the method rationale. The first focus group session is followed by a second session which results in a new version of the method (M_3), carrying the improvements of the second evaluation session. Based on the feedback received in the evaluation session, more cycles can follow. A new focus group session is only scheduled if the evaluation leads to new questions which cannot be answered by analyzing the data. There is no limit set to the number of focus group sessions, and thus there is no limit to the number of method increments. Each session is carried out with a different group of participants, making sure that new data is captured with every cycle. When no new improvements are discovered, or when they do not require a focus group session's feedback, the final version of the method is created (i.e. SPEM).

3.1 Construct validity

Expert interviews were conducted with the goal to gather data on software pattern evaluation, which was the basis of the initial method. An introduction to all underlying theory, the protocol and the goal of the interview was given. The researcher and the interviewee need to have the same understanding of all concepts used during the interview. This was done by stipulating a definition of all concepts together with the interviewee. The researcher gave a definition for a concept based on theory. Afterwards the interviewee was asked if the definition is similar to his understanding of the concept. Together with the interviewee it was decided whether the definition should be changed or the original definition is kept.

Evaluation sessions began with the researcher introducing the theory, goal and protocol of the session. The participants were made aware what was expected of them and how the session would transpire. All activities of both the researcher and participants were explained before commencing with the evaluation session. The definition of an attribute was given by the researcher based on ISO:25010 documentation. Participants were able to share their views on the definition of an attribute in a discussion with fellow participants. Defining attributes and their implications on software quality is an integral part of evaluation sessions. After the session all participants were asked to fill out an evaluation form. The concepts used in this form were introduced by the researcher.

3.2 Internal validity

The aim of evaluation sessions is to improve the initial method. During these sessions the method is performed in a focus group session. After the session all participants fill out an evaluation form. Participants should give their feedback on the method itself, this includes the activities and deliverables used during the session. The performance of the evaluator should not influence the participants in their feedback on the method. For example, a participant can comment on a certain activity being irrelevant because the evaluator did not perform the activity in a way which convinced the participant of the activity's relevance. To prevent this, each evaluation session was performed with the same evaluator and artifacts used by the evaluator. Any differences in the evaluator's style were documented and included in the interpretation of the data.

3.3 External validity

Expert interviews were performed in both a consultancy and a software development company. The problems found with software pattern selection is present in all companies developing software. In the interviews questions were asked about the attributes which might be relevant in software pattern selection. These questions were based on software patterns in general, as an abstract solution rather than an implementation thereof. Therefore, the market the company is in and the types of implementations the

interviewee had experience with do not influence the interview. The results of the interview are generalizable to software development, rather than a specific market related to software development.

3.4 Reliability

Related theory is documented to make the construction of artifacts used in this research traceable by others. The activities and deliverables used during interviews and evaluation sessions are documented and modelled. Questions and answers of the interviews were noted and sent to the interviewee for approval. All data from evaluation sessions was digitized and made available. Both audio and video were recorded during the evaluation sessions. Interpretation of the data is documented and references to the data are included.

3.5 Interview protocol

The interviewee was selected based on accessibility, job description and company size. A qualifying job description is that of software architect with multiple years of experience. Software architect is generally a senior position that is acquired after years of experience in software development. This makes accessibility an important criteria, as there are only few software architects in most companies. A medium to large size company was selected where software architecture has matured and pattern selection is a recurring process. The interview was semi-structured, allowing the interviewee to explain and expand upon each topic. After a brief introduction on the research topic and goal of the interview, the interviewee is presented a list of quality attributes. This list is based on the ISO standard for quality attributes and is used in the domain of software engineering and common basis of software architecture evaluation. It is assumed that the interviewee is familiar with these attributes.

The definition of software pattern evaluation was stipulated together with the interviewee to make sure the outcome of the interview is valid when relating the attributes to the process of software pattern evaluation. For each quality attribute the interviewee was asked if it plays a role in software pattern evaluation. For every question the interviewee was asked to give extensive motivation. Although the goal of the interview was to reduce the list of attributes, it was possible for the entire list to be relevant for software pattern evaluation. Because the quality attributes listed in the ISO standard have been used and validated over time to cover each aspect of software architecture it is not expected that attributes need to be added. However, there was an option for the interviewee to add any attributes he considers missing but relevant. In such case additional motivation was asked to explain its relevancy.

A second interview was conducted to validate the findings of the first. This was done by interviewing a different software architect who works for a different company. The previous results were discussed and validated. All results were documented and served as input for further analysis. After analyzing the results and depending on the validation a decision was made to perform an additional interview.

4. Literature review

4.1 Software Patterns

Patterns and pattern languages originated from architecture (cities, buildings) and were first introduced by Alexander (Alexander, 1977). The concept of his work is that through sharing knowledge on architectural design with patterns, both professionals and non-professionals would be able to practice architecture. He argued that the occupants of buildings would be able to design architecture because they know their requirements. Patterns have been used in a variety of domains and have had a large impact on software development. Patterns have shaped the way software engineers think about software design and how they communicate working design. As software engineering matured in the 80's and object oriented programming became popular, patterns were used informally and in many cases were not yet referred to as patterns (Booch, 1986). They were used unconsciously by developers who communicated their designs through diagrams and documentation. Beck and Cunningham were interested in using patterns for object oriented software design. They followed Alexander's philosophy and let representatives of the user create a user interface design consisting of five patterns (Beck & Cunningham, 1987). In 1994 software patterns were popularized by the publication of the "Gang of Four" book (Gamma, Helm, Johnson, & Vlissides, 1994). Gamma, Helm, Johnson and Vlissides were software engineers that worked at the IBM research department. They joined in a collaborative effort to research and document software patterns. In their work they introduced the term design pattern, a type of pattern which captures object oriented software design. The authors gave examples of design patterns by documenting 23 design patterns using a formal notation. The notation is widely used throughout the software industry to document software patterns and can include (class) diagrams, examples and consequences. The book remains the most influential work in software patterns to this day, well known by many software engineers, especially software architects. The patterns described in the book are often referred to as "Gang of Four patterns" and have been so influential that the knowledge on patterns often does not extend beyond the formal notation and patterns provided in the Gang of Four book (Buschmann, Henney, & Schmidt, 2007).

Research on patterns and pattern documentation has been shared yearly at PLoP conferences since 1994. The works have been published as a collection in later years (Coplien & Schmidt, 1995; Vlissides, Coplien, & Kerth, 1996).

In the early 90's the need for high level software design arose because of increasing complexity and size of software products. This discipline, called software architecture, is based on architectural principles found in regular architecture (Perry & Wolf, 1992).

Since then patterns have also become a part of software architecture, although they were often used informally and unconsciously. Garlan and Shaw documented many commonly used architectural styles (Garlan & Shaw, 1993), which would later be considered architectural patterns (Shaw, 1996).

The role of patterns in software architecture was greatly influenced by the publication of Pattern Oriented Architecture: A System Of Patterns (Buschmann, 1999). Buschman argues that patterns are rarely used standalone and that patterns can be used to design software architecture. While the Gang of Four book focusses on design patterns, Buschman categorizes patterns in idioms, design patterns and architectural patterns. Patterns are described using an elaborate pattern notation which includes examples, guidelines for implementation and variants of the pattern. A new way of thinking about architectural design is proposed by looking at patterns as an element which can be used for constructing architecture with defined properties. It means that a software architect needs to have knowledge on what the desired properties for an architecture are, and then construct a system of interrelated patterns to meet the requirements. A system of patterns is used to describe, classify and relate patterns. Pattern documentation and pattern catalogues describe patterns as standalone solutions, while a system of patterns broadens this view by describing the relationships of patterns and their effects. The work of Buschman has influenced how patterns are perceived, from a view where patterns are used in isolation to solve a particular design problem, to a view where patterns are used together to create a system of patterns, forming a software architecture with defined properties.

The work of Fowler is an example of a collection of patterns which are aimed to create a software architecture (Fowler, 2002). The documented patterns come from Fowler's years of experience in the field. Many of the patterns come from projects that used older technology and programming languages, but Fowler claims that they are generic enough to translate to newer programming languages, such as Java and .NET. He provides patterns categorized by topic, such as 'handling session state in stateless environments'. This work shows that industry solutions have made their way into software architecture, following the efforts of software architecture pioneers.

Patterns are not restricted to object oriented design, but can also be used for interaction design and usability. Borchers experimented with the use of patterns in Human Interaction Design (HCI) (Borchers, 2001). He argues that it is often difficult to express and share interaction design knowledge. In his work pattern languages are presented, which were based on interactive music. Borchers concludes that indeed patterns help capture interaction design knowledge and encourage their use in HCI.

In the work of Zdun an approach to support the selection of patterns based on desired quality attributes and systematic design decisions based on patterns is presented (Zdun, 2007). This approach deals with the problem that pattern documentation is written by different authors and does not provide a formal notation for pattern relationships and their effects on quality goals. This formal notation comes from the creation of pattern language grammar. By scanning pattern documentation a grammar can be created which includes the patterns as words and pattern sequences as sentences. An overview of the grammar provides insight in the pattern variants, sequences and quality goals. Quality goals are scored on a 5-point scale, allowing for negative, neutral and positive scores. Design spaces are used to add rationale to the pattern language grammar. The approach can be used to add in information on quality goals and rationale to existing pattern documentation to aid in the pattern selection process.

Henninger and Correa looked at the current state and challenges of software patterns (Henninger & Correa, 2007). They conclude that many patterns have been documented over the years, but that it has become increasingly difficult to find and select patterns. They support their claims with a survey that shows that 31% of patterns is not electronically available. Even if patterns are available through the web, they can be hard to find because many different pattern forms exist. Pattern forms describe the pattern, pattern language or collection with a set of attributes. They often have a different name for the same attribute, or have non-matching attributes. In order for tools to be developed which facilitate finding the right solution for a design problem, consensus has to be reached on a pattern form. Another problem the authors identify is pattern validation. They state that pattern validation is not part of pattern documentation and that validation and evaluation of patterns can be beneficial to pattern selection and decision making.

Web 2.0 marked a transition from packaged software to software as a service (SaaS) along with many new design principles. O'Reilly looked at how the introduction of Web 2.0 influenced design patterns and business models (O'Reilly, 2007). He explains that Web 2.0 is a buzzword, but that we can find design principles that are linked to it. Examples of this are: packaged software (Web 1.0) vs. services (Web 2.0) and professional content vs. user generated content. Especially the introduction of services have had a profound influence on software patterns. Examples of patterns that are focused on service oriented architecture are SOAP (Simple Object Access Protocol) and REST (Representational State Transfer). Web services can be more dynamic than traditional HTML web-sites by using asynchronous calls in the form of AJAX (Asynchronous JavaScript and XML). At the core, web services are made for reuse and loose coupling which has led to the creation of many new patterns over the years to support asynchronous calls (Garrett, 2005; Gross, 2006).

4.2 Architecture Evaluation

Evaluation is commonly used in software architecture in order to increase quality and decrease cost (Abowd, Bass, Clements, Kazman, & Northrop, 1997). Many evaluation methods for software architecture have been developed and compared in recent years (Babar, Zhu, & Jeffery, 2004). The evaluation should be performed as early as possible in order to prevent large scale changes in later stages of development (Abowd, Bass, Clements, Kazman, & Northrop, 1997). 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 (Hills, Klint, Van Der Storm, & Vinju, 2011) in a quantitative manner, but has focused on the implementation of different patterns and lacks the evaluation of the idea the pattern describes.

SAAM stands for Software Architecture Analysis Method and was created to validate claims on qualities of software architectures (Kazman, Bass, Webb, & Abowd,

1994). The method focusses on describing an architecture and then analyzing its quality attributes. In the original method only modifiability was discussed, but SAAM can be used with other quality attributes as well. The analysis is performed to see if the organization's needs are reflected in the architecture. Therefore SAAM requires that both the organization's needs and software architecture are well described.

Architecture Tradeoff Analysis Method (ATAM) is an evaluation method which allows for analyzing an architecture based on quality attributes (Kazman, Klein, & Clements, 2000). The method allows for analyzing an array of quality attributes and expands on its predecessor SAAM. It is meant as an early analysis, preventing large costs associated with architectural changes. The strength of ATAM is that it takes into account many quality attributes and their tradeoffs, instead of focusing on one quality attribute and ignoring the impact on other quality attributes.

Quality Attribute Workshop (QAW) is a method to evaluate a software architecture based on quality attributes (Barbacci, et al., 2002). It differentiates itself from ATAM by not requiring the existence of a software architecture, thus allowing software architecture evaluation at an earlier stage. Since there is no existing software architecture or developed product, no direct measurements can be performed in the evaluation. QAW involves stakeholders in the evaluation of the software architecture. The stakeholders, together with software architects, discuss and create scenarios which target a specific quality attribute. From these scenarios test cases are created which are analyzed with the software architecture documentation. The analysis might require the creation of documentation in order to address concerns presented in the test case. Using QAW, the software architecture can be evaluated and adapted at an early stage, which can prevent large scale changes to the architecture at a later stage in development. QAW is not a replacement for ATAM, as both evaluate the architecture at different stages.

4.3 Quality attributes

Quality attributes are aspects of quality which allow for the measuring of software quality, also referred to as software quality characteristics (Losavio, Chirinos, Levy, & Ramdane-Cherif, 2003). They have been used in different domains to define what quality is, and how it can be measured (Cech, Kennedy, & Smith, 1960; Triplett, 1969). There are sets of quality attributes throughout the industry and academia, many of which overlap. The biggest difference in sets of quality attributes are sub-attributes, which are related to a main attribute. For example, capacity and resource utilization can be sub-attributes of an attribute called performance efficiency. However, not all sub-attributes relate to the main attribute as strongly and we often see sub-attributes belong to a different attribute in multiple sets of quality attributes. In 1991 an ISO standard for quality attributes in software engineering was published, called ISO:9126 (ISO/IEC, 1991). The standard was created by a joint technical committee of ISO and IEC and consists of six attributes: Functionality; Reliability; Usability; Efficiency; Maintainability and Portability. Over the years there have been many examples of the ISO:9126's use in the industry and academia (Behkamal, Kahani, & Akbari, 2009; Chua & Dyson, 2004; Zeiss, Vega, Schieferdecker, Neukirchen, & Grabowski, 2007). Jung et al. tested the validity of the ISO:9126 standard and conclude that the attributes are valid, but that

some sub-attributes measure the same concept or do not relate to main attribute. For example, they found that security did not relate to the functionality attribute, as it is described in the standard (Jung, Kim, & Chung, 2004).

Because quality attributes allow for the measuring of software quality, they have become a part of software architecture evaluation. Many evaluation methods use quality attributes to define quality, and measure them in software implementation (Barbacci M. R., 1997; Clements, Kazman, & Klein, 2003). Berander et al. tried to define what quality is by studying multiple works in the domain of quality management (Berander, et al., 2005). They found that there are currently two distinct views; quality as conformance to specification and quality as meeting customer needs. Quality models are also mentioned, including ISO:9126. The authors conclude that quality models allow for quality measurements, but that they are a simplified representation of quality. Quality models do not capture all factors that are defined in quality philosophies and should be used in situations where measurements and quantitative data on quality is needed.

4.4 Research implications

In this section we try to give an initial answer to the research questions based on the above literature study.

SQ1: Which attributes are relevant for software pattern evaluation?

There are multiple sets of quality attributes which are used for software architecture evaluation, which is highly related to software pattern evaluation. The **ISO:9126** standard provides a starting point, which has been tested in academic and industrial situations. However, ISO:9126 is a standard for software quality and not specific to software pattern evaluation. Therefore the standard might have to be adapted to only include attributes that patterns can influence.

Patterns could have more attributes relevant for evaluation than ISO:9126 provides. When we look at pattern documentation we see that there is often a section that mentions how the pattern can be **implemented**. If implementation is a factor in documentation, it can be possible that it is an important attribute for pattern selection and pattern evaluation. We cannot get into the specifics of an implementation for evaluation purposes, but the implementation attribute can be adapted to be more general. An example would be ‘ease of implementation’ which can be evaluated in general, while still providing useful information for pattern selection and comparison.

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

To answer this question we look closely at software architecture evaluation and works that attempt to quantify pattern characteristics. Architecture evaluation methods like ATAM involve stakeholders to determine the business drivers and requirements. At a later stage scenarios are created that target a specific quality attribute, which is used for testing whether the requirements for the attribute are satisfied. When we relate this process to pattern evaluation it becomes apparent that business drivers and require-

ments are project specific and cannot be incorporated. However, the concept that stakeholders are involved and a focus on communication can prove valuable for pattern evaluation. In the work of Zdun we can see an approach to assign a score to specific attributes of software patterns. This is done by assigning scores ranging from - - to + +, giving an indication of a positive or negative trait of the pattern. For pattern evaluation purposes we can use a similar approach, although it would be preferable to use a quantitative score range, which would allow for descriptive statistics and comparison. The range of the scores can be determined using expert interviews.

5. Initial Method Construction

In this section the construction process of the initial pattern evaluation method is described. Two expert interviews were conducted following the interview protocol discussed in 3.5 Interview protocol.

5.1 Expert interview 1

The first expert interview took place at a consultancy company with approximately 300 employees. The software architects are a group of five people who work on various projects. The interview was conducted with the principal architect, who had over five years of experience in this role and over 10 years of experience in software development. Questions were divided in two topics, each covering a research question. Nine questions were formulated on the topic of attributes and six questions on the topic of quantification. The interview took approximately 1 hour and 45 minutes to complete and was recorded (Appendix D - Expert interview 1). An introduction was given to explain the context and goal of the interview. Afterward the key concepts, such as software patterns, were stipulated with the interviewee. The first set of questions on the topic of attributes were derived from ISO:9126 with the goal to determine which attributes are relevant for software pattern evaluation and comparison. The following attributes were discussed:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability
- Ease of learning
- Ease of implementation

These attributes find their origin in software quality assessments and software architecture evaluation. However, the differences in software pattern evaluation and software architecture evaluation can mean the standard needs to be adapted. For this reason the attributes were included in the interview, allowing the inclusion or exclusion of attributes in pattern evaluation.

For each of these attributes the architect was asked if it plays a role in pattern selection or comparison. This was purposely done because the goal of the evaluation is to enable pattern comparison. Asking directly which attributes would be relevant for pattern evaluation could be difficult to answer, since evaluating patterns is not usually done by a software architect and would not clarify the goal of the evaluation.

The sub-attributes described in ISO:9126 were not specifically mentioned due to time constraints. They were added to clarify the definition of an attribute in each question. For example, when asking the architect if portability plays a role in pattern selection or comparison, the question was supported by asking if the software can be moved

to other environments (adaptability). This way of questioning makes sure the interviewee takes in mind all aspects of quality described in ISO:9126 and serves as a way to broaden the comprehension of the attribute.

However, not including the relevance of each sub-attribute in the questions, means that the data on which the initial method will be based cannot be entirely accurate on this matter. Therefore the method needs to be evaluated in a later stage, to see which sub-attributes are relevant for pattern evaluation. Although the questions on attributes were closed, each question allowed the interviewee to elaborate and motivate his answer.

5.1.1 Results and interpretation

In this section the results of the interview are discussed and interpreted. Functional suitability plays a role, but this is mostly because interoperability and security are very important aspects of software quality. ISO:9126 describes interoperability and security as sub-attributes of functional suitability. These sub-attributes play such an important role in pattern selection that they should be attributes rather than sub-attributes. On this attribute the architect explains:

"Interoperability and security are very important and should be quality attributes by themselves".

Therefore, ISO:25010 could be a better standard to base pattern evaluation on, as this standard describes both interoperability and security as attributes with their own respective sub-attributes.

This result was the trigger to incorporate the attributes described in ISO:25010 in a second interview. The role of functional suitability in pattern selection is not conclusive from the first interview, because of the difference in the attribute's definition in both standards.

Reliability can play a role in pattern selection, although this is mostly an issue which is handled at infrastructure level. Based on this result, the reliability attribute was added to the initial method. On reliability the architect stated:

"This is sometimes important, but many times this is done at infrastructure level with for instance load balancing."

The fact that it is in most cases handled at infrastructure level is no reason to exclude it, as the method needs to be able to apply to all software patterns whether an attribute plays a frequent part in its selection or not.

Usability plays a role in pattern selection, although some of its aspects are handled by interaction designers or visual artists. This attribute can play a role when selecting patterns for front-end development to create the layout of an application. Asynchronous data retrieval has an impact on usability for which many patterns are available. The architect explained this by stating the following:

"Asynchronous retrieval of data is one area where patterns also influence usability."

Usability was included in the initial version of the evaluation method. *Efficiency*, *maintainability* and *portability* play a role in pattern selection and comparison. *Ease of learning* is an important aspect when selecting patterns as the experience of the development team influences which patterns can be selected. With an experienced team you might select a pattern that has the most benefits while also being complex. *Ease of implementation* plays a role in pattern selection because you want a pattern to be implemented as fast as possible, unless other quality attributes are so important you have to select a pattern that requires more time to implement.

Scenario based evaluation is often used in software architecture evaluation, but is not suitable for evaluating patterns. Scenarios need to be project specific in order to be useful, while for patterns quality attributes play an important role in its selection and an architect is capable of relating the impact a pattern has on software quality to their own project.

It is important to be able to give a negative score to an attribute because a pattern can have a negative impact on software quality. If no negative score is included, architects might tend to see the median as neutral and lower scores as negative. A 5-point or at maximum 10-point scale would be optimal for scoring attributes. Any higher than a 10-point scale would have a negative effect on accuracy.

An architect should be experienced before partaking a pattern evaluation session. The reason for this is that the result of the architecture and all decisions have to be seen in retrospect in order to fully understand them. The differences in experience between architects should be expressed in a weighted score. However, this cannot be captured in any formula and it would be better to use a consensus based score for this. When using a consensus based score it is expected that the architects will share their knowledge in a discussion and that those who have the most experience will also have better or more argumentation for their score. This would be a way to deal with the differences in experience and gain a more accurate score.

When you have only designed architectures but have not been able to see the long term results it is hard to evaluate software patterns. The experience between architects may vary, but at least one architect should have experience using the pattern which is being evaluated. Those who do not have experience using the pattern can still provide meaningful input.

Based on the results of the first expert interview, the following elements were included in the initial method:

- The quality attributes and sub-attributes described in ISO:9126.
- The possibility to assign a negative score.
- An ease of implementation attribute.
- An ease of learning attribute.
- A score range of -5 to +5.
- A discussion activity.
- A consensus based score.

5.2 Expert interview 2

The first interview raised new questions, after which a second interview was scheduled. The second interview was conducted at a large international software company with approximately 1800 employees. The interviewee was a (technical lead) senior developer with over four years of experience in this role and over twenty years of experience in software development. The setup of the interview remained unchanged. The questions on the topic of attributes were altered to include the ISO:25010 standard which was proposed in the first interview. Sub-attributes were included in the interview, although elaboration was not possible due to time constraints. The interview took approximately 2 hours to complete and was recorded.

The first set of questions on the topic of attributes were derived from ISO:25010 with the goal to determine which attributes are relevant for software pattern evaluation and comparison, while the second set of questions aimed to answer how attributes can be quantified.

5.2.1 Results and interpretation

In this section the results of the second interview are discussed and interpreted. Only answers that conflict with the first interview, or answers to questions not present in the first interview are discussed.

The second interview discussed the relevance of quality attributes found in ISO:25010, a standard which is successor to ISO:9126. Functional suitability and its sub-attributes were all found relevant by the interviewee. In the previous interview functional suitability was considered relevant, but the sub-attributes, particularly security, were not. The changes found in the newer standard have made its sub-attributes of functional suitability more relevant for software pattern evaluation. The expectations of the first interviewee are confirmed in the second interview.

Reliability is relevant in software pattern evaluation. The interviewee gives an example of this:

"We use the Mediator pattern to enable unit testing."

The attributes fault tolerance and recoverability were not found relevant, mainly because there were no examples of this at the projects the interviewee had worked on. For recoverability the interviewee mentions:

"We use patterns for recoverability in the system layer, which is not my area of expertise."

Usability is found relevant, with the exception of two sub-attributes, learnability and appropriateness recognisability. The interviewee had no experience with patterns that influenced these two sub-attributes, which does not conclude that they are irrelevant. It appears to be a matter of experience, because it conflicts with the findings of the first interview.

The sub-attribute adaptability of the attribute portability was found irrelevant. This had to do with the fact that the interviewee did not concern himself regularly with adaptability in relation to software architecture. It is also an aspect of quality that is satisfied in the system layer. The interviewee explains:

"This is no day to day business for me, because it is mostly covered in the system layer and because we use things like jQuery."

Compatibility was found relevant while it proved difficult to determine the relevance of the sub-attribute co-existence. Eventually the interviewee decided that co-existance is relevant, but that in his experience it was no relevant factor. The main product of the company is multi-tenant but there is no co-existance of products.

The answers to the questions on quantification were in line with the first interview, with no significant changes. The difference with the first interview is that the interview could not determine whether or not an architect should have experience before partaking in pattern evaluation.

5.3 Implications for pattern evaluation

The previously discussed expert interviews formed the basis of the initial version of the SPEM method (Figure 5 - Initial method (M1)). 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 (ISO/IEC, 2001) and ISO/IEC 25010 (ISO/IEC, 2010) 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 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 (ISO/IEC, 2010).
- 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.

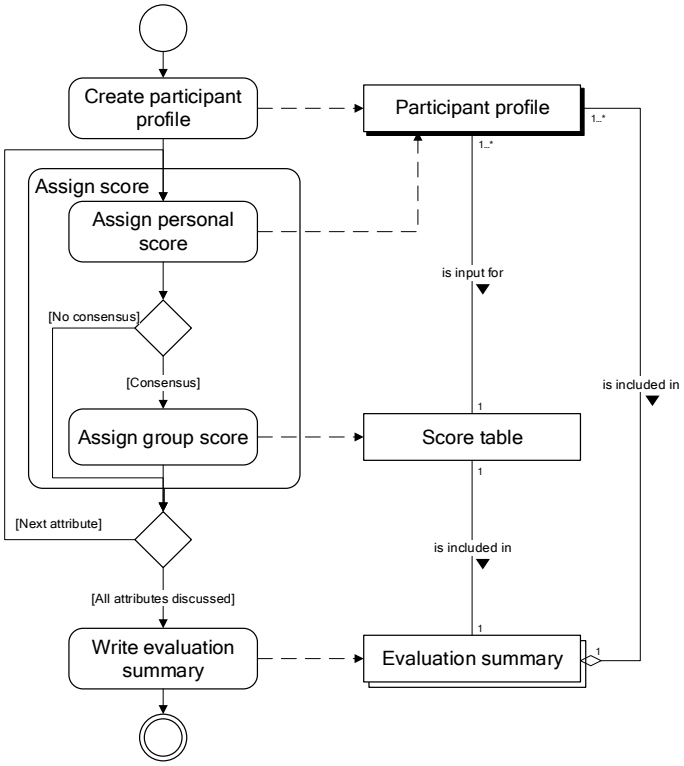


Figure 5 - Initial method (M1)

6. Method evolution

In this section the evolution of the software pattern evaluation method is described. Using a design science approach the initial method was evaluated and improved over several iterations (Figure 6 – Software pattern evaluation method (M2)). 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. The remainder of this section discusses each focus group session and presents the changes which are incorporated in the method.

6.1 Focus group session 1 results and interpretation

During the first focus group session, four software architects participated, each having over nine years of experience in software development (Table 3 - Focus group session participants). During this session the observer pattern was evaluated using the initial version of SPEM. The session took approximately two hours to complete.

Table 3 - Focus group session participants

Description	Participant 1	Participant 2	Participant 3	Participant 4
Experience related to software architecture	12 years	11 years	10 years	10 years
Job description	Principal architect	IT architect	IT architect	IT architect
Has experience using the evaluated pattern	Yes	Yes	Yes	Yes

Method introduction

An introduction was given explaining this research, its relevance and the goal of the evaluation session. The method that would be used was explained, including all activities and deliverables. Finally, an explanation of the session's protocol was given, clarifying who will be performing which activity and in which order. Before continuing to the next step, the participants could ask questions on any of the previously discussed topics.

For some participants the goal of the method was not immediately clear, which led to a discussion among the architects. This discussion was quickly resolved after the principal architect explained his arguments for pattern evaluation.

The goal of the method could be unclear to some because the introduction was intentionally kept as short as possible to focus on the evaluation itself. The participants were briefed on the evaluation session days before the event. In this briefing the goal and context of the evaluation session were explained. However, the evaluation session showed the value of an elaborate introduction, explaining the method with more detail.

Pattern selection

Because the evaluation session was hosted for academic purpose, a pattern needed to be selected for evaluation. A number of patterns were prepared by the evaluator, including high level architectural patterns and lower level design patterns. A restriction of the initial method is that at least one of the participants needs to have experience using the pattern which is being evaluated. Therefore, a pattern was introduced by the evaluator and the participants were asked if they had experience with the pattern. Because of time constraints a pattern was selected that was familiar to all participants. This way less time was spent explaining the pattern and more time was available for the evaluation. After some debate the observer pattern was selected, a pattern which all architects had experience with.

Participant profile creation

Participant profiles were handed out to each participant, after which they were asked to fill in their name, job description, years of experience and experience using the pattern. This process was straight forward and all participants completed it in a few minutes.

Score assignment

For each attribute an introduction was given by the evaluator. A presentation slide pointed out which attribute was being discussed. Sub-attributes were also included on the slide but without a definition. For some sub-attributes the definition was not clear to the participants, which led to discussion. The discussion made the definition clear for all participants, but it also shows that definitions for sub-attributes should have been included in the method. A participant stated after the evaluation that:

"I would add a description for sub-attributes as well and be more explicit about the fact that the evaluation is about the effect of the pattern on the system and not about a specific implementation".

After introducing an attribute the evaluator would give the participants a few minutes time to assign a personal score. The participants had no problem assigning a score within this time frame. At times a participant would assign scores to the attribute and its sub-attributes at once. Although the method was created to go over each attribute and sub-attribute one by one, the participants' tendency to assign scores to them both at once could potentially speed up the score assigning activity. The problem with the initial method is that discussion takes place after each attribute and sub-attribute in order to assign a group score, which restricts the assigning of scores to be done one by one. More sessions are needed to see if discussion and group scores are needed for sub-attributes.

When a personal score was assigned by all participants, the evaluator would initiate a discussion by asking a participant to reveal his score. This process went on without any directing from the evaluator as the participants joined in the discussion. At all times the attribute provoked a discussion, which could take from a few minutes up to twelve minutes. When constructing the initial method it could not be foreseen how architects

would react to a discussion on this topic. The evaluator would need to direct the discussions based on the available time. Directing in this context means that not only does the evaluator initiate the discussion, but also actively reminds the participants of the available time and halts the discussion when no more time is available.

At the end of the discussion the evaluator would ask the participants if they have reached consensus. Either consensus was reached or the discussion would be extended.

Consensus was reached for all attributes except for functional suitability and its sub-attributes. For this attribute it was not possible to assign a score, as functional suitability requires project specific context.

Score interpretation

When we look at the scores assigned in the first focus group session (Appendix F - Focus group session 1 – Score table; Table 4 - FGS1 Score table excerpt) we can see that functional suitability and its sub-attributes did not receive a group score. However, there is an average score because one participant assigned a personal score. After a discussion all participants agreed that functional suitability should not be included in the evaluation, therefore no group score was assigned.

Performance efficiency received a group score of -2, which means the pattern has a negative impact on performance. There is a rather large standard deviation of 1.89 caused by a participant who assigned +2. The participant argued that the pattern could have a positive impact on performance under certain circumstances. The large discrepancy in scores resulted in an average of -0.75. After a discussion consensus was reached on a score of -2, which means that the discussion changed the views of one participant to adopt the arguments of the other three participants. Although some sub-attributes received a different score, they all got assigned -2 after reaching consensus.

Compatibility received a group score of +1 which can be interpreted as mildly positive impact on compatibility. The average showed a similar result with a score of 0.75. The participants agreed that the pattern had no impact or a mildly positive impact on compatibility. After a round of discussion and some examples of implementations, the participants reached consensus on a score of +1. The results of the sub-attributes were close to identical, all receiving a group score of +1, with only a slightly different average score for co-existence (+0,5).

The pattern had no impact on usability and all participants agreed with this standpoint, leaving the average score and group score at 0 (neutral).

Reliability and its sub-attributes received a group score of +1. There was debate on this attribute which is reflected in a high standard deviation of 1.91. The highest score was 4 while the lowest was 0. This was mainly due to the different experiences the participants had with the pattern and how it affected certain implementations. In the discussion the focus shifted to a more general perspective of the pattern and moved away from specific implementations.

The pattern had no impact on security and received a score of 0. Three participants assigned a score of 0, while one participant assigned -2. Similar to reliability, this was due to one participant thinking about a specific implementation. In the discussion all participants agreed that the pattern had no impact on security in general.

Maintainability received a group score of +3, the highest score assigned to any of the quality attributes. The average personal score was +3.5 resembling the group score. All participants quickly agreed that this was the strength of the pattern and that it had a positive influence. The only debate was on how positive this impact was with one participant assigning a personal score of +5 while the others assigned a +3. What stands out is that this is the only attribute where the sub-attributes received a deviating group score.

The participants were divided on analyzability with scores ranging from -4 to +4. A discussion revealed different perspectives on analyzability and consensus was reached on a score of +2.

All participants agreed that the pattern did not have an impact on portability and consensus was quickly reached on a neutral score.

Participants considered the observer pattern easy to implement with an average score of +4.25 and a group score of +4.

Ease of learning received a group score of +3, although there were multiple views on this attribute. Some architects would consider ease of learning to be negative, as time has to be invested to learn the pattern, while others compared how easy the pattern was to learn when compared to other patterns. The latter view was adopted by all participants after a round of discussion.

When looking at the score table we can see that the highest and lowest scores are being avoided. There was no group score of +5 or -5 assigned in the evaluation. A personal score of +5 was assigned eight times, which suggests that discussion and consensus allows for compromise. Four out of ten attributes did not receive a group score and were deemed irrelevant for this particular pattern. The way a participant views the attribute before and after discussion can be different. In some cases a participant would assign a negative personal score and later reach consensus on a positive score. Discussions allow the participants to share their knowledge and learn from each other. The feedback on this activity was positive with a participant stating:

"Discussions during the evaluation session were very useful".

Table 4 - FGS1 Score table excerpt

Pattern name: Observer

Attribute name	Group score	Average	Standard deviation
Functional suitability	**0**	**1,25**	**2,50**
Functional completeness	0	1,25	2,50
Functional correctness	0	0,75	1,50
Functional appropriateness	0	1,25	2,50
Performance efficiency	**-2**	**-0,75**	**1,89**
Time-behavior	-2	-0,50	1,73

Resource utilization	-2	-0,50	1,73
Capacity	-2	0,25	2,06
Compatibility	**1**	**0,75**	0,96
Co-existence	1	0,50	1,00
Interoperability	1	0,75	0,96
Usability	**0**	**0,00**	**0,00**
Appropriateness recognisability	0	0,00	0,00
Learnability	0	0,00	0,00
Operability	0	0,00	0,00
User error protection	0	0,00	0,00
User interface aesthetics	0	0,00	0,00
Accessibility	0	0,00	0,00
Reliability	**1**	**1,50**	**1,91**
Maturity	1	1,00	2,00
Availability	1	1,50	1,91
Fault tolerance	1	1,50	1,91
Recoverability	1	0,50	1,00
Security	**0**	**-0,50**	**1,00**
Confidentiality	0	0,00	0,00
Integrity	0	-0,75	1,50
Non-repudiation	0	0,00	0,00
Accountability	0	-0,75	1,50
Authenticity	0	0,00	0,00
Maintainability	**3**	**3,50**	**1,00**
Modulatiry	4	4,25	0,96
Reusability	2	2,50	1,73
Analysability	2	0,00	3,65
Modifiability	4	4,00	0,82
Testability	3	2,75	2,63
Portability	**0**	**0,25**	**0,50**
Adaptability	0	0,25	0,50
Installability	0	0,25	0,50
Replaceability	0	0,25	0,50
Ease of implementation	4	4,25	0,50
Ease of learning	3	2,25	2,50

Method evaluation

When all scores were assigned, the method was evaluated. The evaluator handed out evaluation forms to all participants. In the evaluation form participants were asked both closed and open questions on each deliverable and activity of the method. The data from these evaluation forms is summarized in Table 5 – Focus group session 1 evaluation data summary.

Table 5 – Focus group session 1 evaluation data summary

Question	P1	P2	P3	P4
Information asked on the participant profile is relevant	Yes	Yes	Yes	Yes
The introduction provides enough information	N/A	N/A	Yes	N/A
The introduction of attributes provides enough understanding	Yes	Yes	Yes	Yes
The score range is sufficient	Yes	Yes	Yes	Yes
The score table includes all relevant score data	Yes	Yes	Yes	Yes
The score table and diagram allow for pattern comparison	No	Yes	Yes	Yes

The information asked on the participant profile was found relevant by all participants. One participant suggested the inclusion of:

"Technology, average size/length of software development".

We chose not to include the suggested fields because technology and size/ length of software development are not always related to experience with patterns. Because 100% of participants answered positively on the relevance of information asked on the participant profile, no changes were made to the deliverable.

When we look at the question on the pattern introduction we see that it was not applicable to 75% of participants. This is caused by the fact that the participants had experience using the pattern, which means that many of them did not learn anything new from the introduction. One participant did find the introduction of the pattern to provide enough information to allow for pattern evaluation. It could be that he was less experienced using the pattern than the other participants. More data is needed to make any statements on the pattern introduction.

All participants thought that the attribute introduction provided enough understanding. It should be noted that all participants had over 10 years of experience in software development, which means that we cannot conclude that the introduction is sufficient for inexperienced participants.

Most participants thought that the score table and diagram allowed for pattern comparison. One participant did not agree and stated:

"Maybe a spider diagram, putting in multiple patterns is better suited".

A spider diagram can be an alternative to the bar diagram used in the evaluation method. However, there is no reason to change the score diagram based on the data provided in the evaluation in which 75% thought the current bar chart allowed for pattern comparison. The feedback will be taken into account for future evaluation sessions.

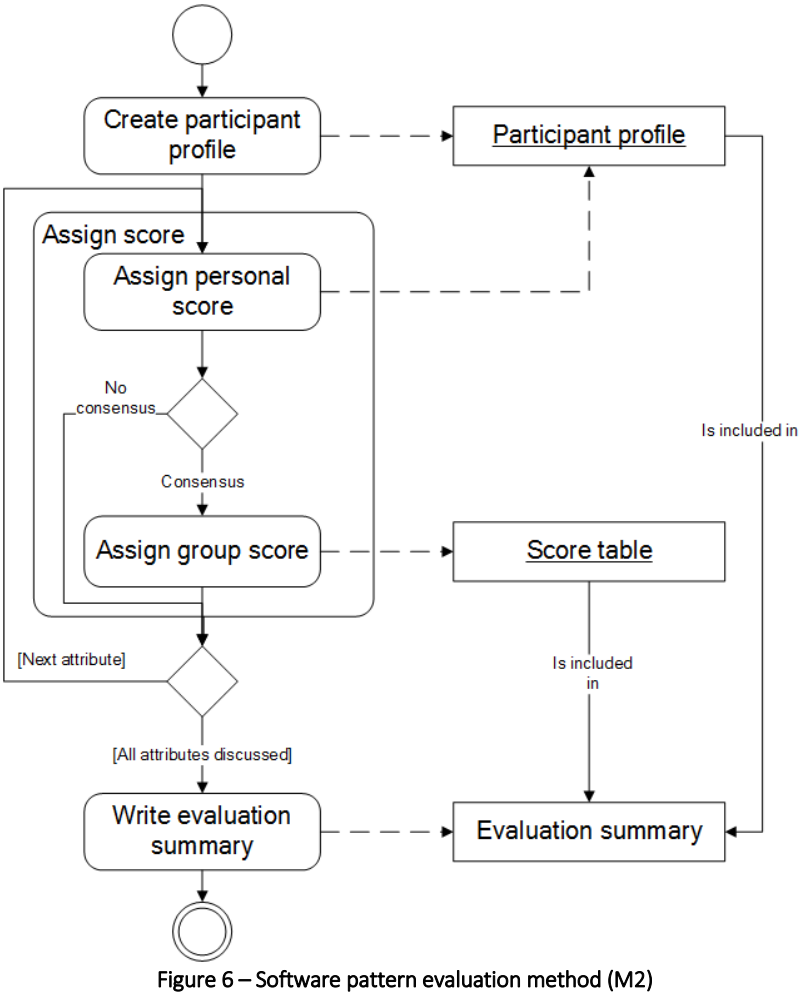


Figure 6 – Software pattern evaluation method (M2)

In Figure 6 – Software pattern evaluation method (M2) the improved method is depicted, it includes the changes incorporated based on the first focus group session.

Changes that are present in processes or deliverables are underlined. The observation, pattern data and evaluation data have led to the following changes:

- **A score range of -3 to +3** — A smaller score range would be sufficient to express the impact a pattern can have on software quality. This score range was suggested by one of the participants. Combined with an analysis of the score table, we can see that extremes (-5 and +5) are being avoided.
- **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.
- **Inclusion of 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.

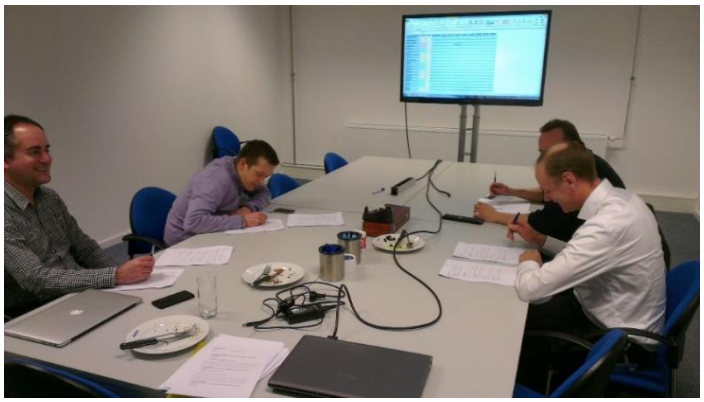


Figure 7 – First method evaluation session

6.2 Focus group session 2 results and interpretation

During the second focus group session, ten software architecture master students participated. 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.
In this session the check point pattern was evaluated using the second (M2) version of SPEM. The session took approximately two hours to complete.

Method introduction
An introduction to the evaluation session was given by the evaluator, explaining key concepts and providing context to the session. Because the group consisted of software architecture master students, the introduction was adapted to include the concept of software patterns and an example of a pattern. This change was excluded from evaluation analysis and cannot be considered part of the method. An aspect that was

changed based on the feedback of the first focus group session is the length of the introduction. For this session the introduction was more elaborate with more examples of the problem statement and what the evaluation's goal is. The introduction consumed 15 minutes after which there was an opportunity for the participants to ask questions. In total the introduction, including questions, did not take more time than it did in the first evaluation session. Although the introduction was more elaborate and longer, the questions did not provoke discussion, making the process shorter.

Pattern selection
It could not be expected that the participants had experience using and selecting patterns, therefore multiple patterns were selected by the evaluator. Depending on the available time, one or more patterns could be evaluated in one session. The check point and limited view pattern were selected. Both are high level security patterns which are easy to comprehend and are used in many well-known software products.

Participant profile creation
A form containing the participant profile was handed out by the evaluator to all participants. The questions on the form were unaltered, although they were not always relevant to the participants. Questions, such as the experience in software architecture were not applicable to many students. Some participants had a few years of experience related to software architecture because they had worked in software development. The feedback on the participant profile's relevance towards students was discarded, as the method is meant to be performed by software architects.

Score assignment
Each attribute was introduced by the evaluator by presenting a short description. Based on the feedback of the first session, sub-attributes were also included in the introduction with a similar description.

After the introduction of an attribute or sub-attribute, the participants assigned a personal score. This process was completed within five minutes for each attribute. However, the participants were not always familiar with the presented attributes. A participant formulated this problem as:

"The definition of the quality attributes need more explaining in some cases. That way a better understanding of the attributes is provided for people who have limited experience with patterns or software architecture".

There was debate on how to interpret the attributes and how scores should be assigned. It is likely that these problems arose because the participants were students who do not possess the same experience evaluating software products as software architects. In this context a participant stated:

"For people without a background in software architecture, there is little information to base an opinion on".

The problem was resolved by allowing the participants to withdraw from scoring if they did not understand the attribute.
When personal scores were assigned, a discussion was initiated by the evaluator. A random participant was asked to reveal his personal score and motivate it. It was expected that students would not have as many arguments as software architects because of a lack of experience. However, as with the first session, each attribute provoked a lengthy discussion. A difference with the first session is that only a few dominated the discussion while others did not engage. It could be the effect of a concept called social loafing, meaning that participants might not agree with what is being said, but choose to keep silent (Latane, Williams, & Harkins, 1979). The evaluator should be aware of this and direct discussions asking questions to those who do not actively participate. One of the participants stressed the importance of this by stating:

"Try to make sure everyone gives his/her opinion".

At the end of the discussion or when no time was left, the evaluator asked if the participants had reached consensus. In the end consensus was always reached, although for some attributes this meant that the discussion had to be extended.
Due to the time spent on discussions and reaching consensus the evaluation could not be completed within a two hour timeframe. Only five out of nine attributes were evaluated, this could be because of the larger focus group of ten participants versus the group of four participants in the first session. Regardless, the method would need to be adapted to work in a much smaller timeframe, otherwise the method becomes impractical and unusable in an industrial setting. A participant defined this problem as:

"The session can be very time consuming if participants are not familiar with patterns or quality attributes".

In order to evaluate a pattern within an hour the evaluator would need to do strict timekeeping. A participant stated the following solution:

"Set a time per attribute, so that every attribute is evaluated within the time of the session".

When time is up, a discussion should be halted whether consensus has been reached or not. Also, participants should not be allowed to discuss previous attributes. Whenever this does happen, the evaluator should direct the discussion towards the attribute that is currently being assessed.
The majority of time is consumed by the discussion of sub-attributes. In order to deal with time constraints the role of sub-attributes in the method would need to be altered.

The focus should be on attributes, while sub-attributes support these, providing better understanding of the attribute and more detailed data. To make it concrete, attributes should be discussed and assigned a group score while sub-attributes would only receive personal scores.

Score interpretation

Nine out of ten participants assigned personal scores. One participant did not assign any scores and is not included in the data analysis. It has to be noted that analysis of the data is not related to the pattern, but to the method itself. For a pattern evaluation this session would not be valid for the following reasons:

- The use of software architecture master students
- Not all attributes were evaluated

However, we can still analyze the behavior and feedback of the participants based on scores, observation and evaluation forms. A complete overview of the scores can be found in (Appendix G - Focus group session 2 – Score table).

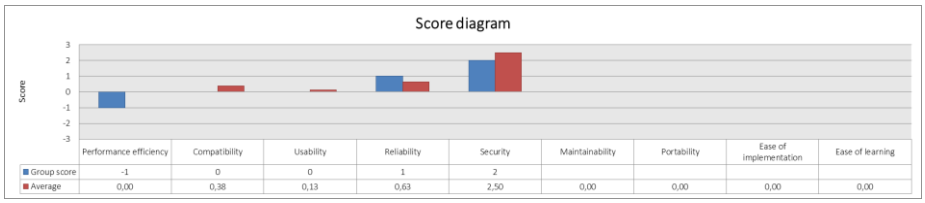


Figure 8 - Focus group session 2 - Score diagram

Consensus for performance efficiency was reached on -1, while personal scores for this attribute were both positive and negative, ranging from -1 to +1. The different perspectives on the attribute were aligned in the discussion. Compatibility and usability both got assigned neutral group score, while again both positive and negative personal scores were assigned. For these attributes some participants requested more explanation on their definition and an example. It is not the role of the evaluator to enter discussion on the definition or view on an attribute or its relation to software quality. Participants should share their views and educate each other in the discussion. This way, a participant might have a different view on an attribute when assigning a personal score than when reaching consensus. It contributes to the value of a group score and knowledge that has been shared in the evaluation session.

Reliability and security both got assigned a positive group score, +1 and +2 respectively. From the start of the discussion it was clear that both these attributes would receive a positive group score. No negative personal scores were assigned either. The attributes were clear to the participants and the evaluated pattern is a security pattern,

which might explain the high score and easily reached consensus on security. There wasn't sufficient time to evaluate any other attributes. The following attributes did not receive a score:

- Maintainability
- Portability
- Ease of implementation
- Ease of learning

Table 6 - FGS2 score table excerpt

Pattern name: Check point

Attribute name	Group score	Average	Standard deviation
Performance efficiency	**-1**	**0,00**	**1,07**
Time-behavior	-1	-1,00	0,00
Resource utilization	-1	-1,00	0,00
Capacity	-1	-0,88	0,35
Compatibility	**0**	**0,38**	1,06
Co-existence	0	0,00	0,00
Interoperability	0	0,00	0,00
Usability	**0**	**0,13**	**0,83**
Appropriateness recognisability	0	0,13	0,83
Learnability	0	0,13	0,83
Operability	0	0,13	0,83
User error protection	0	0,13	0,83
User interface aesthetics	0	0,13	0,83
Accessibility	0	0,13	0,83
Reliability	**1**	**0,63**	**0,52**
Maturity	1	0,63	0,52
Availability	1	0,63	0,52
Fault tolerance	1	0,63	0,52
Recoverability	1	0,63	0,52
Security	**2**	**2,50**	**0,53**
Confidentiality	2	2,50	0,53
Integrity	2	2,50	0,53
Non-repudiation	2	2,50	0,53
Accountability	2	2,50	0,53
Authenticity	2	2,50	0,53

Method evaluation

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. A participant stated that the pattern introduction could be improved:

"Make it more clear by using examples. There was too much discussion over the pattern".

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.

When the focus group session was concluded an evaluation form was handed out to all participants. The data has been summarized in Table 7 - Focus group session 2 evaluation data summary.

Table 7 - Focus group session 2 evaluation data summary

Question	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Information asked on the participant profile is relevant	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No
The pattern introduction provides enough information	Yes	No	No	Yes	No	No	No	No	No	No
The introduction of attributes provides enough understanding	No	No	No	Yes	Yes	No	No	No	No	No
The score range is sufficient	Yes	No	No			Yes	No	Yes	Yes	Yes
The score table includes all relevant score data	Yes	No		Yes	Yes	Yes	Yes	Yes	Yes	No
The score table and diagram allow for pattern comparison	Yes			Yes		No	Yes	Yes	Yes	Yes

When we analyze the evaluation data summary and compare it to the first session we see that the negative feedback has increased **from 4% (first session) to 42%.**

Considering the efforts that were made to improve the method based on the feedback from the first session, one would expect an improvement in positive feedback. It is unlikely that the changes to the method have caused this result, because they had no effect on most of the activities and deliverables found in the evaluation, except for score range.

The increase in negative feedback is most likely caused by the use of software architecture master students as opposed to experienced software architects.

When we look at the individual questions we can see why the feedback has changed over the last two sessions. The information on the participant profile was found relevant by all participants of the first session while it was deemed irrelevant by 30% of participants in the second session. The result is not surprising because the participant profile is created for software architects. Questions, such as, how many years of experience someone has in software architecture, do not apply to software architecture master students. It does not mean that the participant profile needs to be adapted, it only confirms that the profile is not applicable to software architecture master students.

According to 80% of participants the pattern introduction did not provide enough information in order to evaluate the pattern. The reason we see this negative result for the first time in the second session is because the participants had no experience using the pattern, which means they needed a proper introduction in order to understand and evaluate it. This was not the case during the first session where all participants had experience using the pattern and an introduction was not necessary.

It means that the pattern needs to be more thoroughly explained during the introduction in order to enable less experienced participants to partake in the evaluation.

The introduction of attributes did not provide enough understanding according to 80% of participants. The increase compared to the first session, where the negative feedback on this activity was 0%, shows that those who are inexperienced at software architecture most likely need a more elaborate introduction to attributes. Experienced software architects are used to working with quality attributes and need little explanation, this explains why they (in the first session) thought the attribute introduction provided enough understanding. The evaluation method is meant to be accessible to both experienced and inexperienced software architects, which means the method needs to be adapted. The attributes, what they represent and how they should be scored should get more focus in the introduction.

Although 30% of participants did not think the score range was sufficient, the suggested scores ranges were inconsistent, therefore the score range remains unaltered.

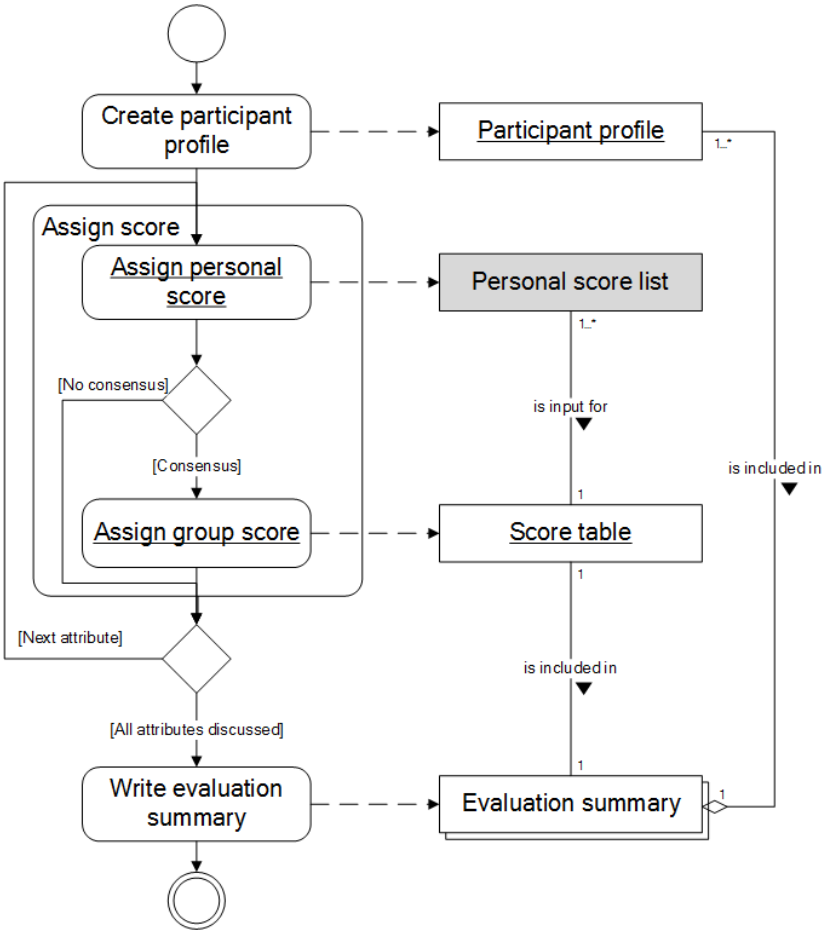


Figure 9 - Software pattern evaluation method (M3)

In Figure 9 - Software pattern evaluation method (M3) the improved method is depicted, it includes the changes incorporated based on the second focus group session. Changes that are present in processes or deliverables are underlined.

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

- **Sub-attribute group scores 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.

- **Personal score list included** — A form containing all attributes and personal scores. Included as a separate deliverable, formerly part of the participant profile.
- **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.

6.3 Focus group session 3 results and interpretation

The third focus group session was performed with different students from the same group as the second focus group session. The session was carried out with eight participants and took approximately one hour to complete. The check point pattern was evaluated, which was selected in the same manner as the second evaluation session. Therefore the selection process will not be described in detail in this section.

Method introduction

The evaluator introduced the method to the participants similarly to the previous evaluation session. The process has been adapted since the last session to focus more on explaining the pattern and keeping short on the theoretical background of the method and software patterns in general. By doing this the understanding of the pattern was improved for the participants, reducing the questions asked during the session. The introduction focused more on the scoring system, in particular the way a score should be assigned. The difference between a quality attribute and a pattern attribute were made clear. Especially the thought process is important when assigning scores, as quality attributes require a participant to think about the effects on software quality. During this session there was no need to explain this more thoroughly as opposed to the previous session. The introduction including questions was completed within 20 minutes.

Score assignment

Scores were only assigned to attributes, excluding sub-attributes. An attribute was introduced by the evaluator with a short description. The participants were given time to assign a personal score. In the previous session there was a debate on how to assign scores, which did not present itself in this session. It is likely that the changes to the introduction discussed in the previous section prevented the debate. This process took between 5 to 10 minutes for each attribute.

After assigning a personal score, a discussion was initiated by the evaluator. The role of the evaluator was more proactive in this session with a focus on strict timekeeping. It was achieved by letting the participants discuss and reminding them of the available time. When no more time was available the discussion was halted by the evaluator, who asked if consensus was reached. When consensus was not reached, the evaluator asked the participants to raise their hand if they had favored a certain score. This way the

evaluator can make a decision to extend the discussion if only a few participants prevent consensus from being reached. At times when participants are divided on a certain attribute the evaluator can decide to halt the discussion and not assign a group score, continuing to the next attribute.

A more proactive role of the evaluator and the exclusion of sub-attributes greatly reduced the time spent on the evaluation. The participants responded positively to the role of the evaluator in the discussion by stating:

"The role of the evaluator was good and allowed us to assign our own scores".
"The group was allowed to reach their own consensus".

Score interpretation

In this section the assigned scores are interpreted. A complete overview of the scores can be found in (Appendix H - Focus group session 3 – Score table).

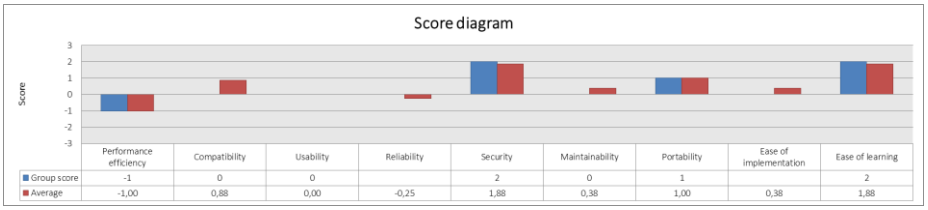


Figure 10 - Focus group session 3 - Score diagram

Performance efficiency received a group score of -1, the same as the average personal score. There was not much discussion, because all participant agreed on a negative or neutral impact. For compatibility personal scores were assigned between -1 and +3, resulting in a high standard deviation of 1.25. There was no debate on the attribute itself, only on the effects of the pattern on this aspect of software quality. Consensus was quickly reached, assigning a neutral score. Usability had a similar result, with both negative and positive personal assigned scores. The final group score was 0 (neutral). For reliability consensus was not reached because one participant could not be convinced by the other participants. This shows that a participant can at times solely prevent consensus and is not overcome by any social pressure by his peers. The pattern was found to have a positive impact on security by six out of eight participants. The other two assigned a neutral score, while the average score was +1.88 and consensus was reached on +2. An attribute that stood out was ease of implementation, for which no consensus was reached. It was caused by three participants who thought the score should be negative and four participants who thought the score should be positive. The discussion was not extended because too many participants had a different view on this attribute.

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.

Table 8 - FGS3 Score table excerpt

Pattern name:	Check point	
Attribute name	Group score	Average
Performance efficiency	-1	-1,00
Compatibility	0	0,88
Usability	0	0,00
Reliability		-0,25
Security	2	1,88
Maintainability	0	0,38
Portability	1	1,00
Ease of implementation		0,38
Ease of learning	2	1,88

Figure 11 - Method evaluation session at Utrecht University

Method evaluation

At the end of the session the method was evaluated by means of an evaluation form. The questions on the form remain unchanged from focus group session 1 and 2. The data of the evaluation has been summarized in Table 9.

.

Table 9 - Focus group session 3 evaluation data summary

Question	P1	P2	P3	P4	P5	P6	P7	P8
Information asked on the participant profile is relevant	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
The introduction provides enough information	Yes	No	Yes	Yes	Yes	Yes	No	Yes
The introduction of attributes provides enough understanding	No	Yes	No	Yes	Yes	No	Yes	Yes
The score range is sufficient	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
The score table includes all relevant score data	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
The score table and diagram allow for pattern comparison	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Overall we see that compared to the second focus group session, the responses to the third session were more positive. In the second session 42% of the answers were negative, indicating that many activities and deliverables needed to be adapted. The changes to the method resulted in 15% negative answers to the questions found in the method evaluation.

*Compared to the second session, negative feedback found in the evaluation results has **decreased by 64%**.*

When we look at the individual questions we can see what attributes to this decrease. All participants thought that the information asked on the participant profile was relevant. In the previous session 30% did not agree with the statement, which can be explained by a better introduction. Because the session was performed with students it is important that they are told the participant profile is targeted towards software architects. Although this was explained in the first session, the larger emphasis on this aspect of the introduction can explain why 100% now agrees with the statement.

The more elaborate pattern introduction was beneficial to the participants.

*The number of participants who though that the introduction did not provide enough information was **reduced by 68%.***

This result confirms the observation that during this session there were no discussions on the pattern itself. No questions were asked about the pattern, although there was a difference in how it was perceived by some participants. The way a pattern is viewed by participants should not be considered part of the introduction. Aligning views can effectively be done during the discussion activity.

What attributes represent and how they should be scored was elaborated on during the attribute introduction. The method was improved to focus on the impact that quality attributes have on software quality and how patterns can influence these.

*The evaluation data also reflects this improvement reducing the negative feedback on the attribute introduction activity from 80% (second session) to 38%, a **53% reduction.***

The score range was no concern in previous session although a minority considered the score range to be insufficient. There was no consistency in the score ranges suggested by the participants. Although no changes were made to the score range, negative feedback on the -3 to +3 range was reduced to 13%. An explanation for this can be the added focus on how scores should be assigned in the introduction activity.

The score table does not have to be adapted based on the evaluation data. Only 13% thought the score table did not include all relevant data, but the suggested changes, like including standard deviation, were already present in the score table.

The score table and score diagram allow for pattern comparison, which is a consistent result from all three evaluation sessions.

*When comparing the last two evaluation sessions the number of **unanswered** questions in the evaluation forms has been reduced **from 10% (second session) to 0%.***

It could be that the participants were more confident answering the questions, because all activities and deliverables were elaborately discussed during the introduction.

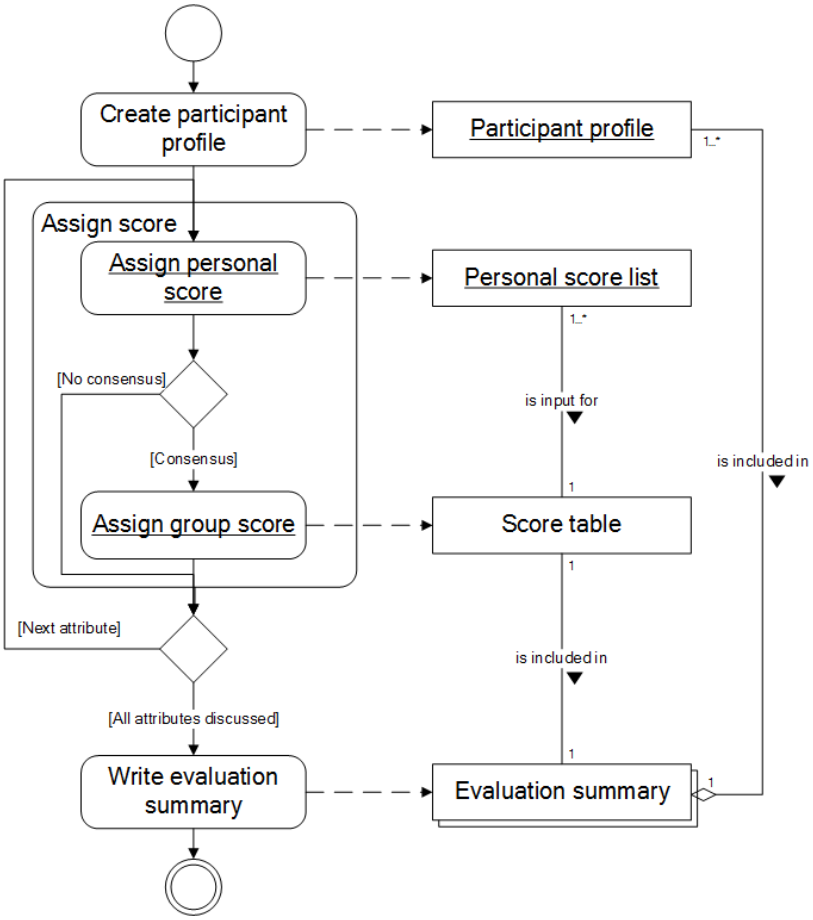


Figure 12 - Software pattern evaluation method (M4)

In Figure 12 - Software pattern evaluation method (M4) the improved method is depicted, it includes the changes incorporated based on the third focus group session. Changes that are present in processes or deliverables are underlined.

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.

7. SPEM – Software pattern evaluation method

In this section the final version of the pattern evaluation method called SPEM (Software Pattern Evaluation Method) is presented. This method is the result of a literature study, expert interviews and three focus group evaluation sessions. First, the method is discussed in general, describing the stakeholders, activities and deliverables. Two subsections explain the method from the perspective of the evaluator and the participant, going more into detail on their individual activities.

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 (ISO/IEC, 2010). 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.

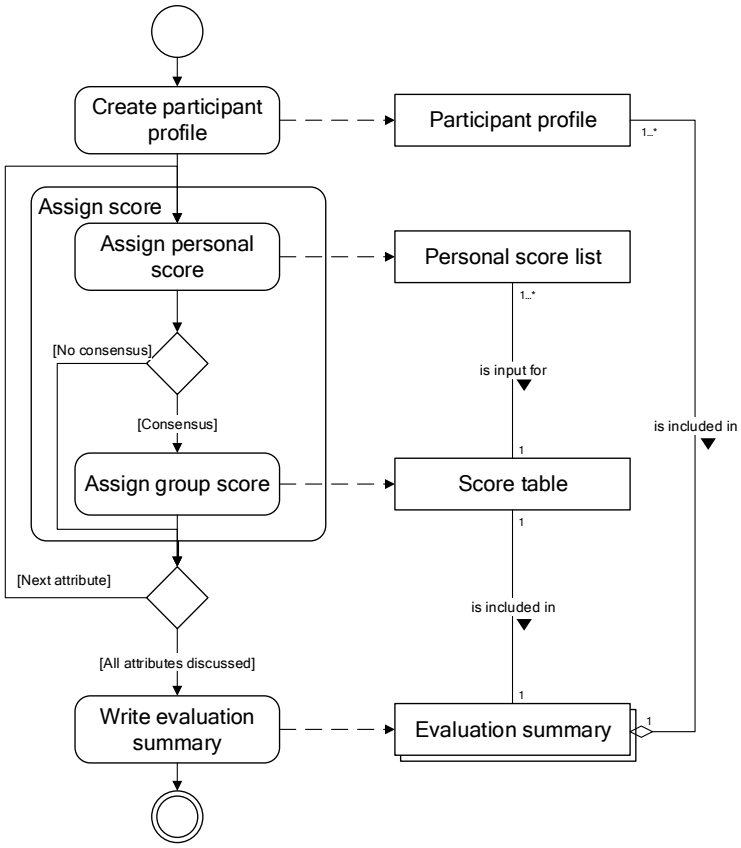


Figure 13 - SPEM: Software Pattern Evaluation Method

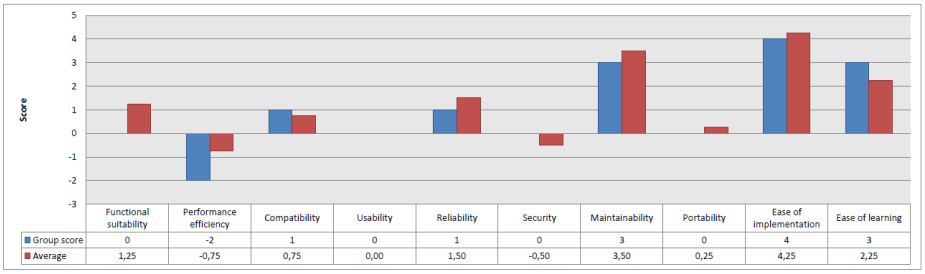
Table 10 - SPEM activity table

Activity	Sub-acitivity	Description
Create participant profile		A PARTICIPANT PROFILE is created for each participant.
Assign score	Assign personal score	A personal score is assigned to an attribute and corresponding sub-attributes. The score is assigned on the PERSONAL SCORE LIST.
	Assign group score	A group score is assigned to an attribute after discussion and reaching consensus.
Write evaluation summary		All data is gathered, meta-data is added and the score table is filled out.

Table 11 - SPEM concept table

Concept	Description
Personal profile	A form containing fields for the participant's name, job description and experience,
Personal score list	A list containing all attributes and sub-attributes with input fields for assigning scores.
Score table	A table containing all attributes and sub-attributes with columns used for assigning personal scores and group scores.

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 14).



	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

Figure 14 - SPEM evaluation summary (observer pattern)

SPEM consists of four activities and four deliverables, as shown in Figure 13. The first activity focuses on creating participant profiles. 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 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 (Figure 15 - Score table excerpt). 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.

Pattern name:	Observer						
Attribute name	Group score	Average	Standard deviation	Participant 1	Participant 2	Participant 3	Participant 4
Functional suitability	0	**1,25**	**2,50**	0	5	0	0
Functional completeness	0	1,25	2,50	0	5	0	0
Functional correctness	0	0,75	1,50	0	3	0	0
Functional appropriateness	0	1,25	2,50	0	5	0	0
Performance efficiency	**-2**	**-0,75**	**1,89**	-2	-1	2	-2
Time-behavior	-2	-0,50	1,73	-1	-1	2	-2
Resource utilization	-2	-0,50	1,73	-1	-1	2	-2
Capacity	-2	0,25	2,06	2	-1	2	-2

Figure 15 - Score table excerpt

In the process **assign group score** 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.

a. Role of the evaluator

The evaluator leads the SPEM session by directing discussions, introducing concepts and writing an evaluation summary (Figure 16 - Evaluator activity diagram). It is required that an evaluator has understanding of SPEM, software patterns and attributes to an extent which allows the evaluator to introduce and explain each of the concepts. Although this role can be filled in by software architects or developers, the role is not restricted by any set of job descriptions. Anyone with the required knowledge can fill in the role of evaluator.

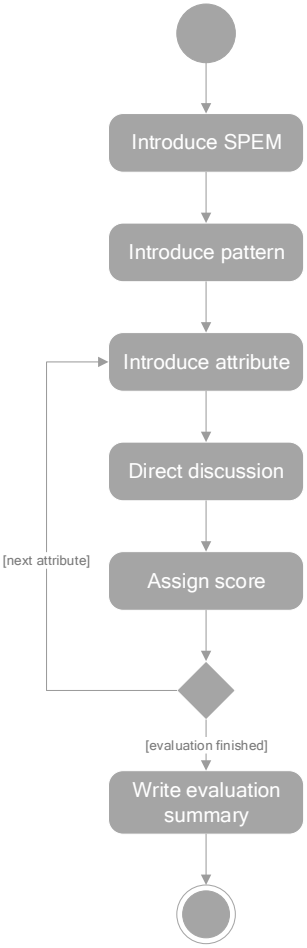


Figure 16 - Evaluator activity diagram

There are six main activities carried out by the evaluator. An introduction to SPEM is given in the form of a short presentation. The goal is to create understanding of SPEM among participants, allowing them to participate in the evaluation. It is important that the difference between quality attributes and pattern attributes is made clear. Participants need to be made aware how scores should be assigned to different types of attributes, as each type of attribute require a different thought process. For quality attributes this means that a participant should a score based on the impact the pattern has on software quality as criteria. For pattern attributes the score can be assigned based the pattern's characteristics as criteria. This is a crucial part of the evaluation and it is the responsibility of the evaluator to make sure all participants are able to assign scores.

The evaluator gives an explanation of the pattern which is being evaluated. This is done briefly without going into the specifics of implementation. The problem the pattern is trying to solve and its solution should be explained in general. For the ease of understanding, a depiction of the pattern's solution can be included. After this activity all participants should have basic knowledge of the pattern.

When the participants are ready to assign scores, the evaluator introduces an attribute and then directs a discussion. This process is repeated until all attributes have been assigned a score. The introduction of attributes has been included for two reasons;

1. To create understanding of the attribute.
2. To define the attribute, preventing different interpretations among participants.

The definition of an attribute is also present on the personal score list.

When participants have assigned a score to the attribute and corresponding sub-attributes, the evaluator initiates a discussion. The goal of the discussion is to reach consensus and assign a group score to the attribute (excluding sub-attributes). The role of the evaluator in this process is to direct the discussion. After opening the discussion the evaluator asks a random participant to reveal his score and give motivation for the score. Other participants can raise their hand if they want to enter the discussion. The evaluator directs the discussion by giving the floor to participants who want to enter the discussion. If participants do not make apparent they want to enter the discussion, the evaluator can initiate by asking if any participants have a different score. When scores no longer differ or when all motivation and arguments have been shared, it is an indication that consensus has been reached. In this case the evaluator halts the discussion and asks if consensus has been reached by asking if a score which was dominant during the session can be agreed upon. When all participants agree, consensus has been reached and a group score can be assigned. If consensus is not reached the evaluator can decide to extend the discussion. This can help in cases where few participants disagree with the others.

Discussion is a time consuming process which can take anywhere from a few minutes to tens of minutes. It depends on the available time whether discussions can go on, therefore timekeeping is an important responsibility of the evaluator. It can mean that a discussion is halted prematurely by the evaluator and that consensus may not be reached.

The final activity of the evaluator is to write an evaluation summary. In this summary all deliverables are collected. At this stage the evaluator fills out the score table, adding all personal scores and visualizing the data in the form of a diagram. A form is included containing the date, evaluator's name, pattern name and number of participants. There is an optional field for threats to validity which could not be expressed in the deliverables.

An evaluation summary is a deliverable which concludes and summarizes a SPEM session. It provides information on when the evaluation has taken place and who participated. Without having to examine the data, the evaluation summary can inform the reader of the evaluation's reliability and validity.

The summary can help someone decide to use the session's data, discard it or redo the evaluation. For the last option it is important that it is transparent who has participated in the evaluation and what threats to validity might have occurred. This information could improve any future evaluation sessions of the same pattern.

b. Role of the participant

A participant enters discussion, assigns personal scores and creates a participant profile (Figure 17 - Participant activity diagram). This role can be taken by developers or software architects. It is required that a participant has knowledge of software patterns and experience using them. Any knowledge on SPEM is not required as it is introduced by the evaluator. It is desirable that the participant has used the pattern that is being evaluated in a SPEM session, it is not mandatory for every participant. At least one participant should have experience using the evaluated pattern. This requirement can be validated in the participant profile, where experience with the software pattern is noted.

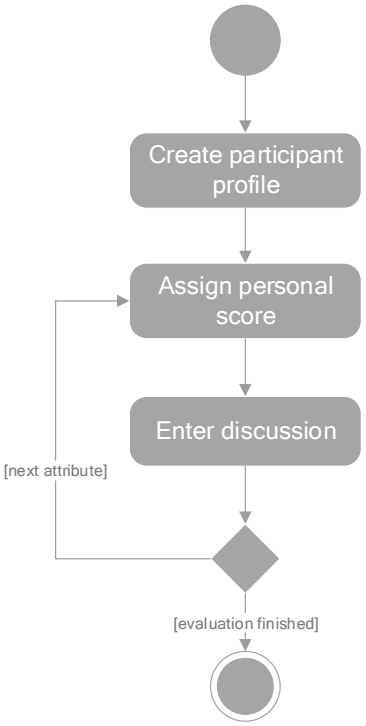


Figure 17 - Participant activity diagram

The first activity of a participant is to create a participant profile. This is done by filling out a form containing personal information. There is a field for the name of the participant, which is used to link the personal scores to a participant. The participant's job description is asked in order to validate if a person has met the requirements.

The second activity of a participant involves assigning personal scores. A score is assigned to an attribute and corresponding sub-attributes. The participant notes the score on a form, the personal score list. This activity ensures that each attribute and sub-attribute is assigned a score. The scores represent the impact a software pattern has on software quality or a characteristic of the pattern itself. It allows for descriptive statistics, providing averages and standard deviations. However, personal scores are not weighted, each score from each participant is weighted the same. Scores can be less accurate when participants with varying degrees of experience participate in a SPEM session. To cope with this problem, group scores were introduced.

The final activity of a participant is to enter discussion. A participant shares his score and motivates it. During the discussion a participant can respond to others, share knowledge and gain a better understanding of the pattern. It is assumed that participants that have the most experience also have a stronger case when defending their score. At the end of the discussion all participants work towards consensus. Consensus does not have to be reached and any participant is free to stick to their personal score. When consensus is reached a group score is assigned otherwise it is left blank.

8. 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 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.

SPEM can be used as a survey rather than focus group sessions with a few alterations. The participant profile and personal score list would need to be included in the survey along with a description of the evaluation itself. The process of assigning group scores would need to be removed from the evaluation.

This would allow a large amount of data to be collected, consisting of personal scores assigned to attributes. It would expand the possibilities of statistical analysis of the data beyond descriptive statistics. Before altering SPEM as a method based on surveys, it is important to value the importance of discussion and consensus. This can be done by

looking at the difference between group scores and average scores. The data from this study does not allow for testing if the difference between these two scores is significant. To test this, another study would be required using a larger population.

8.1 Industrial use

SPEM gains its relevance from a problem found in software development, namely selecting a software pattern when multiple alternatives exist. Experience is the basis of software pattern selection, but is not always available. SPEM tries to capture this knowledge to aid inexperienced software architects in the pattern selection process. Pattern selection can have effects throughout software development as design decisions cannot easily be changed further in development, therefore this decision can have large implications for a project. Any tool which can help a software architect make better decisions represents value for a company. However, SPEM requires that a company invests time in an evaluation session. It means that pattern evaluation becomes a business decision, where the value needs to outweigh the effort. SPEM is most valuable if the pattern selection involves a decision between patterns which are crucial for a project. For example, in a mobile banking application security patterns can have such implications that an evaluation session is viable. *The decision to use SPEM depends on the potential value gained from selecting one pattern over another.* This value can be different for each project and pattern selection process.

Each evaluation summary can be stored in a knowledge base, which makes SPEM results easily accessible. Because accessing the knowledge base is not as time consuming as an evaluation session, the threshold for using the knowledge base is lower.

8.2 Academic use

This study provides knowledge in the domain of software engineering by introducing a method to evaluate software patterns. The method enables gathering data on software patterns through evaluation sessions. Evaluation summaries provide understanding of software patterns and their relation to software quality. The summaries can be accumulated and stored in a knowledge base. Research using SPEM can add results to the knowledge base, making it accessible for others. As more evaluation results are gathered, the knowledge on software patterns is expanded. The academic value of a SPEM knowledge base lies beyond the domain of software patterns, as evaluation summaries contain data on software quality. The link between software patterns and software quality can be the basis of future work. The knowledge base opens up the possibility for statistical analysis. For example, it would be possible to see what impact security patterns have on performance or maintainability in general. When examining large sets of data the trade-offs in software quality by using certain types of patterns can become apparent.

SPEM is restricted to the evaluation of software patterns in general. Project specific pattern evaluation is not possible and would require more research. It is unknown what categorization of projects would enable comparison while providing more detailed data.

Without categorization a projects name or description can be too detailed, making it difficult to find other evaluations to compare with. There are no problems when two patterns are evaluated at the same company for a specific project. However, the results would not be as valuable to academia as a more general evaluation method such as SPEM.

An area in which SPEM can possible be improved is the method of data gathering. Focus group sessions are restricted to small number of participants. A way to make the data more reliable is by using a larger population. To do this, a different method of data gathering is needed. A survey could increase the amount of data at the cost of group scores. The focus group sessions allow for discussion and assigning group scores. Research is needed to find out if the difference between average scores and group scores is significant. If the difference is not significant, SPEM could be adapted to use a survey as method of data gathering.

8.3 SPEM's position in software architecture

In this section we position SPEM within the domain of software architecture. It is important to evaluate the software architecture at an early stage. This can be done by performing a Quality Attribute Workshop (QAW), which does not require a (partially) developed product. However, a draft of the architecture is required, which means that patterns should have been selected at this stage.

The output of the workshop provides us with data on:

- Quality attributes that are crucial to the project.
- The part of the architecture that needs to be adapted.

When a test case targeting a specific quality attribute is not passed, the architecture needs to be adapted. In many cases this means that new patterns need to be selected to satisfy the requirements. However, the functional requirements remain the same, so the selection process focuses on selecting alternative patterns that have desired effects on software quality. From the early architectural evaluation we know which quality attributes are crucial, which can serve as input for pattern selection.

At this point the architect faces the scenarios discussed in (1 - Problem statement), which depending on the scenario, determines the next step. If the architect has experience using the alternative patterns, no further evaluation is needed. However, if the architect is inexperienced with one or more of the alternative patterns, a SPEM session can be of value. Such session could provide the necessary data a pattern's impact on software quality in the form of scores. The scores can then be related to the crucial quality attributes found in QAW.

Although SPEM is ideally used at an early stage of architectural design, it can also be used in conjunction with ATAM. QAW and ATAM use a similar approach towards software architecture evaluation, both using stakeholders, quality attributes, scenarios and test cases. The difference lies in how test cases are performed and analyzed. While QAW relies on documentation and modelling, ATAM allows for quality attribute analysis on a (partially) developed software product. The output of ATAM can be the input for a SPEM session. The risks identified by ATAM can be used to alter the architecture,

likely leading to the selection of alternative patterns. For this type of pattern selection SPEM can be a valuable tool. The data provided by ATAM allows for pattern selection based on quality attributes, as quality requirement elicitation has taken place. Therefore it is ideal that alternative pattern selection contains data on the impact on quality attributes provided by SPEM.

SPEM allows for the selection of software patterns based on quality requirements elicited in software architecture evaluation.

8.4 Advantages of using SPEM

In this section the advantages of using SPEM are discussed in general, and compared to traditional advice. The advantages associated with software evaluation, such as **improved software quality** and **cost reduction**, also apply to SPEM. Software patterns have a direct effect on software quality, consequently selecting one pattern over another can influence the system's quality. SPEM helps understand what quality attributes are affected by a pattern and to what degree, which in turn can help software architects during the pattern selection process. SPEM evaluations can be performed at an early stage in development and do not require any form of implementation. Therefore it can contribute at the early stages of creating a software architecture.

By selecting patterns that satisfy quality requirements at an early stage, large costs associated with architectural changes can be prevented.

SPEM allows for more efficient pattern selection by eliminating the need to study all candidate patterns. When multiple patterns solve the same problem, it becomes important to see which will be the best fit for the project. By looking at SPEM evaluation summaries an architect can quickly assess which patterns possibly satisfy the project's quality requirements. SPEM provides quantitative visualized data, which allows for direct comparison of patterns.

The result is that less patterns have to be studied in depth, reducing the time needed to select patterns.

The results of SPEM sessions can be stored and shared, making it an effective tool for communication. The sharing of knowledge on patterns and their effect on software quality can further improve efficiency in the pattern selection process. Using existing SPEM evaluation summaries eliminates the need to perform evaluation sessions.

SPEM allows knowledge of software architects on patterns to be captured in a way that resembles advice from coworkers. There are some key advantages by using SPEM over traditional advice:

- Group scores
- Moderated discussions
- Consensus

- Formal processes and deliverables

Consensus based group scores force software architects to express themselves quantitatively and prevents a flood of qualitative data on experiences with pattern implementation. This way SPEM provides essential data on a pattern's effects on quality attributes, which allows for quick assessment.

Discussions are moderated which helps SPEM to be performed in a reasonable time frame, which can be adapted to satisfy business needs. SPEM uses formal processes and deliverables which ensures that the output can be compared to any other SPEM evaluation regardless of who participated.

The focus of all SPEM processes and deliverables is on ***improving software quality by sharing knowledge*** *and* ***reducing costs by aiding in the software pattern selection process****, making it a valuable tool for both academic and industrial purpose.*

9. Discussion

9.1 Interpreting SPEM evaluation summaries

SPEM allows for evaluating software patterns as an abstract concept, rather than an implementation thereof. This means that the scores of a SPEM session should not be interpreted as an absolute measurement. The scores are a relative representation of a pattern's effects on software quality and its own characteristics. SPEM aims to quantify and objectify the knowledge and experience of a group of experts. Attributes are quantified by assigning scores and objectified by doing this with a group. It should be noted that the limited size of a focus group does not allow for objective evaluation. When performing multiple evaluation sessions on the same pattern, scores may vary. The goal of SPEM is to be as objective as possible within the boundaries of focus group sessions.

The results of a SPEM session can be interpreted as advice from coworkers to aid a software architect when deciding which patterns to use.

The value in the evaluation result does not lie in the exact score but rather the combination and their positive or negative values.

SPEM results need context and knowledge in order to be interpreted. It is necessary that someone has read the pattern documentation or has a general understanding of the pattern before interpreting an evaluation summary. Knowledge of software development is needed to relate the scores to a project and future implementation. Relying solely an evaluation summary for pattern selection does not always improve decision making. The complexities of implementation and knowledge thereof is key in understanding and interpreting evaluation results.

Patterns have been compared using expert interviews is in the past. There are a few problems with this way of data collection. It can be argued that expert interviews are not suited as a method of data collection for pattern comparisons. Software architecture relies on craftsmanship, accumulated knowledge based on previous implementations. The projects in which a pattern has been encountered can shape the view of an architect on that particular pattern. The importance of experience and different implementations found in projects make interviews particularly subjective when it comes to pattern comparison. We can also see that in the evaluations that were performed in this study, the scores and comments varied between participants and arguments for scores were based on individual experience with pattern implementation.

If we want a more complete picture of a pattern's effect on quality attributes, we need multiple architects to share their experiences on implementation in order to get a more objective view. The shared experiences of a group of architects can help us to see what the effects of patterns are in general. If the goal is to compare patterns to teach others what their effects are, then we should try to stay as objective and general as possible. Otherwise the comparison is only potentially interesting for those who intend to use a similar implementation and work with similar projects.

9.2 SPEM over conventional software architecture discussions

Software architectural knowledge is shared in formal evaluations and documentation, but also in informal meetings and discussions among architects. One could argue that the same results that SPEM provides can be obtained through informal discussions. However, such discussions are not an alternative to SPEM, as they lack the *direction*, *moderation*, *consensus*, *quantification* and *artifacts* that SPEM provides. Without the direction and moderation provided by SPEM, a discussion can become uncontrolled and lose focus. After a SPEM evaluation session an architect said:

"We have regular discussions, but they quickly resort in endless debate on technicalities. We don't focus on any particular result and do not easily agree with each other".

This is also the observation from the evaluation sessions that were hosted with software architect participants. Architects can easily divert from the main subject as they passionately share their knowledge on technical details and past experiences. An evaluator that directs, keeps time, and moderates discussion helps to focus and reach consensus.

A regular discussion allows for knowledge to be shared, but not stored. Therefore we need a person that records the session and creates artifacts, which is the role of the evaluator in SPEM. This is an added value because it means that the knowledge is not only shared with those who participate in the SPEM session, but also with the industry. The results can be viewed for immediate or future use.

SPEM discussions have a very specific goal, namely to assign consensus based scores.

It is not the discussion that is central to software pattern comparison, it is the combined quantified and visualized output that allows for comparison.

Comparing software patterns in a regular discussion does not contain the structure or output that allows the architect to compare based on quality requirements. A regular discussion might be easier to set up, but is not suited for pattern evaluation and comparison.

9.3 Restrictions

The role of evaluator is an important aspect of SPEM evaluation. The reason not to restrict this role to software architects or developers comes from the fact that an evaluator does not use his experience to assign scores. An evaluator uses knowledge of SPEM and basic knowledge of software patterns. It is possible that a software architecture student, junior developer or business consultant could lead a SPEM evaluation. It is desirable that an evaluator does not have any direct relationship with the participants and does not share the same experience. The evaluator should be as objective as possible when directing discussions. Any presumed notions of what a score might or should be can be an interference.

At least one participant should have experience using the pattern when performing a SPEM session. The value of a SPEM evaluation session lies in the experience of the participants. Although participants can comprehend the evaluated pattern by listening to a pattern introduction and reading documentation, they would lack experience of using the pattern. It is not required that all participants have experience using the pattern, because there is room for discussion and questions during the discussion activity. It also means that there is always at least one participant that can share his experience with the pattern and can answer questions.

SPEM sessions make use of focus group sessions, which means that all participants need to participate at the same time. Optimally the participants are in the same room, but a digital conference could also be possible. The use of a focus group session also means that a group of participants is needed, rather than a single individual. It is recommended that a minimum of four participants attend the session, in order to allow of discussions and reliable results. Of course a lower of participants could be possible, although it will affect the reliability of evaluation in a negative way. There is no maximum number of participants, although we have learned from this study that 10 participants can lead to a successful evaluation. The restriction for the maximum number of participants is therefore up to the available time that has been reserved for the session, as more participants often leads to longer discussions and requires more moderation.

10.Conclusion and future work

In this section we try to answer the research questions which were defined in section 0.

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

SPEM is an objective software pattern evaluation method which can be used to compare patterns. It is used to evaluate relevant attributes of patterns and software quality 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).

SQ1: Which attributes are relevant for software 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:

- Performance efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

Two attributes deduced from pattern documentation have also been added:

- Ease of implementation
- Ease of learning

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

The attributes provided by SQ1 can be quantified in a manner that allows for comparison 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, summarized and visualized in the evaluation summary, allowing for pattern comparison.

Future work

A general evaluation of software patterns is a first step in pattern evaluation. In order to gain more accurate data from an evaluation session, the environment in which the pattern will be used needs to be included. A study is required to adapt SPEM into taking the project environment into account. A different thought process is needed to assign scores, as in SPEM these score are assigned in general, without a project in mind. A concern for this type of evaluation is how data can become more accurate while still allowing for comparison. As projects are described with more detail, it becomes less likely to find evaluations with the same description, which could lead to invalid comparisons. Of course when we evaluate for a specific project, new possibilities arise, such as involving the project's stakeholders in the evaluation. Such evaluation can complement an early software architecture evaluation such as QAW, which involves stakeholders and identifies critical non-function requirements. The next step could be to select new patterns based on a similar early evaluation before implementation.

Another way to make the data of SPEM more accurate is by including the architectural style in the method. By evaluating patterns with a certain architectural style in mind, the data that is produced can include whether a pattern fits the architectural style. Architectural styles might have to be classified in order to make the result more comparable. A way to classify architectural styles has been provided by (Shaw & Clements, 1997).

The focus group sessions performed in this research shaped the final version of the SPEM method. The next step is to perform SPEM sessions to gather data and produce results that allow for software pattern comparisons. In this regard the final method presented in this study, is the first version to be used in an industrial setting and real-world scenarios. The method might be further improved from the experience gained by future evaluation sessions using SPEM, because this study cannot cover all scenarios found in the software industry. 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 academic and industrial purpose.

11.Acknowledgements

I would like to thank Jaap Kabbedijk for his guidance and the many discussions we've had. Secondly, I would like to thank Slinger Jansen and Jan Martijn van der Werf for acting as second supervisor and providing me with feedback. I could not have done this study without the cooperation of Exact software and Info Support. I thank Raymond Brookman from Info Support and Leo van Houwelingen from Exact software for providing me with an interview and sharing their knowledge. I'd like to thank Peter Hendriks, Sander Molendijk, Mark Rexwinkel and Raymond Brookman for participating in a focus group evaluation session. And finally I would like to thank Utrecht University for helping me host focus group evaluation sessions and providing a focus group session room. There are too many names for me to mention in this section, but I would like to thank the master students who participated in the multiple evaluation sessions, which proved invaluable for this study.

References

Abowd, G., Bass, L., Clements, P., Kazman, R., & Northrop, L. (1997). *Recommended Best Industrial Practice for Software Architecture Evaluation.* Tech. rep., DTIC Document.

Alexander, C. a. (1977). Pattern languages. *Center for Environmental Structure*.

Babar, M. A., & Gorton, I. (2007). A tool for managing software architecture knowledge. *Proceedings of ICSE Workshop on Sharing and Reusing Architectural Knowledge (SHARK)*, (pp. 11-17).

Babar, M. A., Zhu, L., & Jeffery, R. (2004). A framework for classifying and comparing software architecture evaluation methods. *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, (pp. 309-318).

Barbacci, M. R. (1997). *Principles for Evaluating the Quality Attributes of a Software Architecture.* DTIC Document.

Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., & Wood, W. (2002). Quality attribute workshops.

Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture in Practice, 2/E.* Pearson Education India.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice.* Addison-Wesley Professional.

Beck, K., & Cunningham, W. (1987). Using pattern languages for object-oriented programs.

Beck, K., Crocker, R., Meszaros, G., Vlissides, J., Coplien, J. O., Dominick, L., & Paulisch, F. (1996). Industrial experience with design patterns. *Proceedings of the 18th international conference on Software engineering*, (pp. 103-114).

Beck, K., Crocker, R., Meszaros, G., Vlissides, J., Coplien, J. O., Dominick, L., & Paulisch, F. (1996). Industrial experience with design patterns. *Proceedings of the 18th international conference on Software engineering*, (pp. 103-114).

Behkamal, B., Kahani, M., & Akbari, M. K. (2009). Customizing ISO 9126 quality model for evaluation of B2B applications. *Information and software technology*, 599-609.

Berander, P., Damm, L.-O., Eriksson, J., Gorschek, T., Henningsson, K., Jonsson, P., . . . Ronkko, K. (2005). Software quality attributes and trade-offs. *Blekinge Institute of Technology*.

Booch, G. (1986). Object-oriented development. *Software Engineering, IEEE Transactions on*, 211-221.

Booch, G. (2005). On creating a handbook of software architecture. *Conference on Object Oriented Programming Systems Languages and Applications: Companion to the 20 th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, *16*, pp. 8-8.

Borchers, J. O. (2001). A pattern approach to interaction design. *AI & SOCIETY*, 359-376.

Bosch, J., & Molin, P. (1999). Software architecture design: evaluation and transformation. *Engineering of Computer-Based Systems, 1999. Proceedings. ECBS'99. IEEE Conference and Workshop* (pp. 4--10). IEEE.

Brito e Abreu, F., & Melo, W. (1996). Evaluating the impact of object-oriented design on software quality. *Software Metrics Symposium, 1996., Proceedings of the 3rd International*, (pp. 90--99).

Buschmann, F. (1999). *Pattern oriented software architecture: a system of patters.* Ashish Raut.

Buschmann, F., Henney, K., & Schimdt, D. (2007). *Pattern-oriented Software Architecture: On Patterns and Pattern Language.* John Wiley & Sons.

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). Past, present, and future trends in software patterns. *Software, IEEE, 24*(4), 31-37.

Cech, M. Y., Kennedy, R., & Smith, J. a. (1960). Variation in some wood quality attributes of one-year-old Black Cottonwood [Populus trichocarpa. *Tappi*, 957-9.

Chua, B. B., & Dyson, L. E. (2004). Applying the ISO 9126 model to the evaluation of an elearning system. *Proc. of ASCILITE*, (pp. 5-8).

Clements, P., Kazman, R., & Klein, M. (2003). *Evaluating software architectures.*

Coplien, J. O., & Schmidt, D. C. (1995). *Pattern languages of program design.* ACM Press/Addison-Wesley Publishing Co.

Fowler, M. (2002). *Patterns of enterprise application architecture.* Addison-Wesley Longman Publishing Co., Inc.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: elements of reusable object-oriented software.* Pearson Education.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Gang of four. *GoF). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, ISBN, 20163361*.

Garlan, D. (2000). Software architecture: a roadmap. *Proceedings of the Conference on the Future of Software Engineering* (pp. 91-101). ACM.

Garlan, D., & Shaw, M. (1993). An introduction to software architecture. *Advances in software engineering and knowledge engineering*, 1-40.

Garrett, J. J. (2005). Ajax: A new approach to web applications.

Gross, C. (2006). *Ajax patterns and best practices.* Springer.

Henninger, S., & Correa, V. (2007). Software pattern communities: Current practices and challenges. *Proceedings of the 14th Conference on Pattern Languages of Programs* (p. 14). ACM.

Hills, M., Klint, P., Van Der Storm, T., & Vinju, J. (2011). A case of visitor versus interpreter pattern. In *Objects, Models, Components, Patterns* (pp. 228-243). Springer.

Höst, M., Regnell, B., & Wohlin, C. (2000). Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering, 5*(3), 201-214.

ISO/IEC. (1991). ISO 9126/ISO, IEC (Hrsg.): International Standard ISO/IEC 9126: Information Technology-Software Product Evaluation. *Quality Characteristics and Guidelines for their use}*, 12-15.

ISO/IEC. (2001). *ISO/IEC 9126. Software engineering -- Product quality.*

ISO/IEC. (2010). *{ISO/IEC 25010. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models}.*

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

Jung, H.-W., Kim, S.-G., & Chung, C.-S. (2004). Measuring software product quality: A survey of ISO/IEC 9126. *IEEE software*, 88-92.

Kabbedijk, J., Galster, M., & Jansen, S. (2012). Focus Group Report: Evaluating the Consequences of Applying Architectural Patterns. *Proceedings of the 17th European conference on Pattern Languages of Programs (EuroPLoP 2012).*

Kazman, R., Bass, L., Webb, M., & Abowd, G. (1994). SAAM: A method for analyzing the properties of software architectures. *Proceedings of the 16th international conference on Software engineering* (pp. 81--90). IEEE Computer Society Press.

Kazman, R., Klein, M., & Clements, P. (2000). *ATAM: Method for architecture evaluation.* DTIC Document.

Khare, R., & Taylor, R. (2004). Extending the representational state transfer (rest) architectural style for decentralized systems. *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on* (pp. 428-437). IEEE.

Klein, M. H., Kazman, R., Bass, L., Carriere, J., Barbacci, M., & Lipson, H. (1999). Attribute-based architecture styles. *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, (pp. 225-244).

Latane, B., Williams, K., & Harkins, S. (1979). Many hands make light the work: The causes and consequences of social loafing. *Journal of personality and social psychology*, 822.

Losavio, F., Chirinos, L., Levy, N., & Ramdane-Cherif, A. (2003). Quality characteristics for software architecture. *Journal of Object Technology*, 133-150.

Meszaros, G. a. (1998). A pattern language for pattern writing. *Pattern languages of program design*, 529--574.

O'Reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & Strategies*.

Perry, D., & Wolf, A. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 40-52.

Polillo, R. (2012). Quality models for web [2.0] sites: a methodological approach and a proposal. In *Current Trends in Web Engineering* (pp. 251-265). Springer.

Roberts, D. J. (1996). Evolving frameworks: A pattern language for developing object-oriented frameworks. *Pattern languages of program design*, 471--486.

Schmidt, D. C. (1995). Using design patterns to develop reusable object-oriented communication software. *Communications of the ACM, 38*(10), 65-74.

Shaw, M. (1995). Patterns for software architectures. *Pattern languages of program design*.

Shaw, M. (1996). Some patterns for software architectures. *Pattern languages of program design*, 255-269.

Shaw, M., & Clements, P. (1997). A field guide to boxology: Preliminary classification of architectural styles for software systems. *Computer Software and Applications Conference, 1997. COMPSAC'97. Proceedings., The Twenty-First Annual International* (pp. 6-13). IEEE.

Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline.* Prentice Hall Englewood Cliffs.

Triplett, J. E. (1969). Automobiles and hedonic quality measurement. *The Journal of Political Economy*, 408-417.

Tyree, J., & Akerman, A. (2005). Architecture decisions: Demystifying architecture. *Software, IEEE, 22*(2), 19-27.

Vlissides, J. M., Coplien, J. O., & Kerth, N. L. (1996). *Pattern languages of program design 2.* Addison-Wesley Longman Publishing Co., Inc.

Yoder, J., & Barcalow, J. (1998). Architectural patterns for enabling application security. *Urbana*, 61801.

Zdun, U. (2007). Systematic pattern selection using pattern language grammars and design space analysis. *Software: Practice and Experience*, 983--1016.

Zeiss, B., Vega, D., Schieferdecker, I., Neukirchen, H., & Grabowski, J. (2007). Applying the iso 9126 quality model to test specifications. *Software Engineering*, 231-242.

Appendix

A. Participant profile

Participant profile
Name: _____
Job description*: _____
Years of experience*: _____
*Relevant to software architecture
Pattern name: _____
I have experience using the pattern which is being evaluated in this session.
Yes ☐ No ☐

Personal score list

1. **Performance efficiency**
   The degree to which the software product provides appropriate performance, relative to the amount of resources used, under stated conditions.

   -3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
   Highly negative impact Neutral Highly positive impact

   1.1 Time-behavior
   The degree to which the software product provides appropriate response and processing times and throughput rates when performing its function, under stated conditions.

   -3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
   Highly negative impact Neutral Highly positive impact

   1.2 Resource utilization
   The degree to which the software product uses appropriate amounts and types of resources when the software performs its function under stated conditions.

   -3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
   Highly negative impact Neutral Highly positive impact

1.3 Capacity

The extent to which the maximum limits of a product or system parameter meets the requirements.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

---

2. Compatibility

The ability of two or more software components to exchange information and/or to perform their required functions while sharing the same hardware or software environment.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

2.1 Co-existence

The degree to which the software product can co-exist with other independent software in a common environment sharing common resources without any detrimental impacts.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

2.2 Interoperability

The degree to which the software product can be cooperatively operable with one or more other software products.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

---

3. Usability

The degree to which the software product can be understood, learned, used and attractive to the user, when used under specified conditions.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

3.1 **Appropriateness recognisability**

The degree to which the software product enables users to recognise whether the software is appropriate for their needs.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

3.2 Learnability

The degree to which the software product enables users to learn its application.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

3.3 Operability

The degree to which the software product makes it easy for users to operate and control it.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

3.4 User error protection

The degree to which the software product provides help when users need assistance.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

3.5 **User interface aesthetics**

The degree to which the software product is attractive to the user.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

3.6 Accessibility

The degree of operability of the software product for users with specified disabilities.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

4. Reliability

The degree to which the software product can maintain a specified level of performance when used under specified conditions.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
Highly negative impact Neutral Highly positive impact

4.1 Maturity

The degree to which a system, product or component complies to reliability needs under normal operating conditions.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
Highly negative impact Neutral Highly positive impact

4.2 Availability

The degree to which a software component is operational and available when required for use.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
Highly negative impact Neutral Highly positive impact

4.3 Fault tolerance

The degree to which the software product can maintain a specified level of performance in cases of software faults or of infringement of its specified interface.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
Highly negative impact Neutral Highly positive impact

4.4 Recoverability

The degree to which the software product can re-establish a specified level of performance and recover the data directly affected in the case of a failure.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐
Highly negative impact Neutral Highly positive impact

5. **Security**

The protection of system items from accidental or malicious access, use, modification, destruction, or disclosure.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

5.1 Confidentiality

The degree to which the software product provides protection from unauthorized disclosure of data or information, whether accidental or deliberate.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

5.2 Integrity

The degree to which the accuracy and completeness of assets are safeguarded.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

5.3 **Non-repudiation**

The degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

5.4 Accountability

The degree to which the actions of an entity can be traced uniquely to the entity.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

5.5 Authenticity

The degree to which the identity of a subject or resource can be proved to be the one claimed.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

---

6. Maintainability

The 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.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

6.1 Modularity

The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

6.2 Reusability

The degree to which an asset can be used in more than one software system, or in building other assets.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

6.3 Analysability

The degree to which the software product can be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

6.4 **Modifiability**

The degree to which the software product enables a specified modification to be implemented. The ease with which a software product can be modified.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

6.5 Testability

The degree to which the software product enables modified software to be validated.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

---

7. Portability

The ease with which a system or component can be transferred from one hardware or software environment to another.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

7.1 Adaptability

The degree to which the software product can be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

7.2 **Installability**

The degree to which the software product can be successfully installed and uninstalled in a specified environment.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

7.3 Replaceability

The extent to which a product can replace another specific software product, with the same purpose in the same environment.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Highly negative impact Neutral Highly positive impact

---

8. Ease of implementation

The ease with which the software pattern can be implemented.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Difficult to implement Neutral Easy to implement

---

9. Ease of learning

The ease with which the software pattern can learned in order to implement it.

-3 ☐ -2 ☐ -1 ☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐

Difficult to learn Neutral Easy to learn

B. Score table

Pattern name:

Attribute name	Group score	Average	Standard deviation	Participant 1	Participant 2	Participant 3	Participant 4
Performance efficiency		"					
Time-behavior		"	"				
Resource utilization		"	"				
Capacity		"	"				
Compatibility		"	"				
Co-existence		"	"				
Interoperability		"	"				
Usability		"	"				
Appropriateness recognisability		"	"				
Learnability		"	"				
Operability		"	"				
User error protection		"	"				
User interface aesthetics		"	"				
Accessibility		"	"				
Reliability		"	"				
Maturity		"	"				
Availability		"	"				
Fault tolerance		"	"				
Recoverability		"	"				
Security		"	"				
Confidentiality		"	"				
Integrity		"	"				
Non-repudiation		"	"				
Accountability		"	"				
Authenticity		"	"				

Maintainability		"	"				
Modulatiry		"	"				
Reusability		"	"				
Analysability		"	"				
Modifiability		"	"				
Testability		"	"				
Portability		"	"				
Adaptability		"	"				
Installability		"	"				
Replaceability		"	"				
Ease of implementation		"	"				
Ease of learning		"	"				

C. Evaluation summary

SPEM evaluation summary

Date: _____
Evaluator: _____
Pattern: _____
Number of participants: _____

Threats to validity:

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

D. Expert interview 1

Date: 18-9-2013

Interviewer profile

First name: René

Last name: van Donselaar

Interviewee profile

First name: Raimond

Last name: Brookman

Company name: Info Support

Job description: Software Architect

Years of experience: 5 years

Research question

RQ1) Which **quality attributes and pattern characteristics** play a role in the software pattern evaluation process?

Introduction

In many cases software architecture evaluation is performed by looking at qualities the product or system should have. Since software patterns are a part of modern software architecture, it is expected that quality attributes like those found in the ISO/IEC 9126 standard also play a role in software pattern evaluation. The next series of questions involve each quality attribute of the ISO/IEC 9126 standard. It is important to note that software pattern evaluation means the evaluation of a pattern based on its documentation and not a specific implementation.

Q1) **Functionality** – Is functionality an aspect of quality that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- The software can perform the tasks required? (***Suitability***)
- Is the result as expected? (***Accurateness***)
- Can the system interact with another system? (***Interoperability***)
- Does the software prevent unauthorized access? (***Security***)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Interoperability and security are very important and should be quality attributes by themselves.

Q2) **Reliability** – Is reliability an aspect of quality that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- Have most of the faults in the software been eliminated over time? (***Maturity***)
- Is the software capable of handling errors? (***Fault tolerance***)
- Can the software resume working and restore lost data after failure? (***Recoverability***)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, "yes" should be answered. Only when an attribute is never used in these cases, "no" should be answered.)

Elaboration:
This is sometimes important, but many times this is done at infrastructure level with for instance load balancing.

Q3) **Usability** – Is usability an aspect of quality that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- Does the user comprehend how to use the system easily? (***Understandability***)
- Can the user learn to use the system easily? (***Learnability***)
- Can the user use the system without much effort? (***Operability***)
- Does the interface look good? (***Attractiveness***)

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Sometimes, in many cases this is important for interaction design. However sometimes patterns are used to create the layout or to maintain consistency. Asynchronous retrieval of data is one area where pattens also influence usability. Attractiveness is not important as it is handled by designers rather than architects.

Q4) **Efficiency** – Is efficiency an aspect of quality that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- How quickly does the system respond? (***Time behavior***)
- Does the system utilize resources efficiently? (***Resource utilization***)

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
In almost all cases important. In very many cases this is a crucial selection criteria.

Q5) **Maintainability** – Is maintainability an aspect of quality that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- Can faults be easily diagnosed? (***Analyzability***)
- Can the software be easily modified? (***Changeability***)
- Can the software continue functioning if changes are made? (***Stability***)
- Can the software be tested easily? (***Testability***)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:

Klik hier als u tekst wilt invoeren.

Q6) **Portability** – Is portability an aspect of quality that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- Can the software be moved to other environments? (***Adaptability***)
- Can the software be installed easily? (***Installability***)
- Does the software comply with portability standards? (***Conformance***)
- Can the software easily replace other software? (***Replaceability***)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
In many cases this is important with certain constraints. Platform independency can be an important when comparing patterns.

Q7) **Ease of learning** – Is ease of learning a pattern characteristic that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- How long does it take for a developer to master the pattern?
- Is the pattern easy to understand?

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
The experience of the team that is going to work with the pattern is important when comparing patterns. With a very experienced team you might select the pattern that has the most benefits while also being complex.

Q8) **Ease of implementation** – Is ease of implementation a pattern characteristic that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- How long it takes to implement the pattern?

- What difficulties can be expected when implementing the pattern?

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
In almost all cases it is important that a pattern can be implemented as fast as possible, unless other quality attributes are so important that you have to select a pattern that requires more time to implement.

Q9) **Dependency** – Is dependency a pattern characteristic that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- Does the pattern need other patterns to function?
- Does the pattern belong to a group of patterns?

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
This is important because if certain patterns have already been used, the evaluated pattern can build upon the existing structure.

Research question

RQ2) How can attributes of software patterns be **quantified** in a manner that **allows for comparison**?

Introduction

In order for quality attributes to be comparable, they need to be quantified. If the same scale and quantification method is used, the results of the evaluation can be compared with one another. This is done by quantifying the results of an evaluation session where multiple experts give their scores. Evaluation of software architectures is common practice in the domain of software architecture. The characteristics of several popular assessment methods have been deduced and form the basis of the following questions. Based on these questions a quality attribute quantification method for software patterns is created.

Q1) **SAAM** – Is it important to give a value to quality attributes based on multiple scenarios?

It is possible that a quality attribute would be valued differently under different circumstances. For example the impact on performance of a software product by a software pattern can be very different for a large scale application then it is for a smaller project. Including these scenarios might give the software architect more detailed information on the software pattern.

Answer: Yes ☐ No ☒

Elaboration:
Scenarios can be important for software architecture evaluation, but not for pattern evaluation. In most cases it are the quaty attributes which are used for evaluation and it is not hard for a software architect to interpret those and relate them to their project.

Q2) Should it be possible to give a **negative value** to a quality attribute?

The influence of a software pattern on the software product can be negative at times. Each pattern has strengths and weaknesses that affect quality attributes. It might be important for a software architect to know that the implementation of a certain software pattern has a negative influence on the quality of software.

Answer: Yes ☒ No ☐

Elaboration:
Patterns can have negative effects on software and it is handy to include that in the evaluation. Otherwise one would probably stay in the middle range for a neutral score and interpret a lower score as negative.

Q3) Should **trade-offs** between quality attributes be included?

In many cases there are certain trade-offs between quality attributes. A pattern which adds more security can do so at the cost of performance. Although all quality attributes are valued, the specific trade-offs could also be included in the evaluation. Adding information on trade-offs would give additional information on quality attributes and their relationships.

Answer: Yes ☒ No ☐

Elaboration:
Adding trade-offs can further explain the scores. Also some scores really ask for more explanation, especially when they are unexpected.

Q4) At what **scale** should attributes be measured?

The scale determines the level of detail at which quality attributes are valued. A larger scale can provide more detail, but only if the evaluation yields meaningful information that can be quantified on such scale. In other words, we want to provide the highest level of detail at which the evaluation can still provide meaningful results.

Answer: 3-point scale ☐ 5-point scale ☒ 10-point scale ☒ 100-point scale ☐

Elaboration:
A minimum of 5 and a maximum of 10. A higher scale would not add much, as it is not possible to be that accurate.

Q5) Should the **experience** of a software architect be **weighted**?

When an evaluation of software patterns is performed, it is done by a group of experts. The experience of an expert accounts for much of the knowledge on software patterns and their quality attributes. When the results of an evaluation session are quantified the experience of each software architect can be weighted in the score to enable a higher influence by those who have more experience.

Answer: Yes ☒ No ☐

Elaboration:
An architect should at least have 2 years of experience before evaluating. The reason for this is that the result of the architecture and all decisions have to be seen in retrospect in order to

fully understand them. When you have only designed architectures but have not been able to see the long term results it is hard to evaluate software patterns. It should be possible for less experienced architects partake in the evaluation if there is at least one experienced architect. The expectation is that the most experienced architect has good reasoning for his score and will automatically have most influence in the discussion.

Q6) Should the software architect have **experience using the pattern**?

When evaluating a pattern the knowledge of multiple experts is used as input. However, it might not be possible for an expert to perform a thorough evaluation based on general understanding of patterns without having experience with the pattern that is being evaluated. Can a pattern be evaluated based on a description of the pattern without having experience using it?

Answer: Yes ☒ No ☐

Elaboration:
It is important for an architect to have experience using the pattern that is being evaluated. Although an architect who hasn't used the pattern can also give meaningful input to the evaluation, at least one architect should have experience using the pattern or the evaluation would not have enough value.

General remarks or notes

The new ISO 25010 standard would be better as a basis for software pattern evaluation. The main reason being that certain sub-attributes

are so important (security and interoperability) that they should be a quality attribute.

It would be better to gain consensus among the focus group when determining the score for a quality attribute. This means that the group can discuss the score and then determine one score which all evaluators can agree upon. This process also gives the more experienced architects a chance to explain their viewpoint and convince others. Therefore it is not needed to give weighted scores, because it is expected that more experienced architects also have more arguments to explain their score.

If some architects would give a very discrepant score and cannot be convinced, it should be possible to remove those outliers and value the more experienced architects over those with less experience.

E. Expert interview 2

Date: 19-9-2013

Interviewer profile

First name: René

Last name: van Donselaar

Interviewee profile

First name: Leo

Last name: van Houwelingen

Company name: Exact

Job description: Software engineer senior / Technical lead

Years of experience: 3 years in this role (23 years in total)

Research question

RQ1) Which quality attributes and pattern characteristics play a role in the software pattern evaluation process?

Introduction

In many cases software architecture evaluation is performed by looking at qualities the product or system should have. Since software patterns are a part of modern software architecture, it is expected that quality attributes like those found in the ISO/IEC 25010 standard also play a role in software pattern evaluation. The next series of questions involve each quality attribute of the ISO/IEC 25010 standard. It is important to note that software pattern evaluation means the evaluation of a pattern based on its documentation and not a specific implementation.

Q1) **Functional suitability** – Is functional suitability (geschiktheid) an aspect of quality that plays a role when *selecting* or *comparing* software patterns?
(De mate waarin een softwareproduct of computersysteem functies levert die voldoen aan de uitgesproken en veronderstelde behoeften, bij gebruik onder gespecificeerde condities.)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:

Klik hier als u tekst wilt invoeren.

When selecting or comparing software patterns does a software architect take into account:

Functionele compleetheid (Functional completeness)

De mate waarin de set van functies alle gespecificeerde taken en gebruikersdoelen ondersteunen.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Functionele correctheid (Functional correctness)
De mate waarin een softwareproduct of computersysteem de juiste resultaten met de benodigde nauwkeurigheid beschikbaar stelt.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Functionele toepasselijkheid (Functional appropriateness)
De mate waarin de functies bijdragen aan het behalen van specifieke taken en doelen.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Q2) **Reliability** – Is reliability (betrouwbaarheid) an aspect of quality that plays a role when *selecting* or *comparing* software patterns?
(De mate waarin een systeem, product of component gespecificeerde functies uitvoert onder gespecificeerde condities gedurende een gespecificeerde hoeveelheid tijd.)

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or

comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Bijv: We gebruiken Mediator pattern om unit testing mogelijk te maken.

When selecting or comparing software patterns does a software architect take into account:

Volwassenheid (Maturity)
De mate waarin een systeem, product of component aan betrouwbaarheidsbehoeften voldoet onder normale werkomstandigheden.

Answer: Yes ☒ No ☐
Elaboration:

Beschikbaarheid (Availability)
De mate waarin een systeem, product of component operationeel en toegankelijk is wanneer men het wil gebruiken.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Foutbestendigheid (Fault tolerance)
De mate waarin een systeem, product of component werkt zoals bedoeld ondanks de aanwezigheid van hard- of softwarefouten.

Answer: Yes ☐ No ☒
Elaboration:
Ik kan me zo geen voorbeeld herinneren.

Herstelbaarheid (Recoverability)

De mate waarin het product of systeem, in geval van een onderbreking of bij een fout, de direct betrokken gegevens kan herstellen en het systeem in de gewenste staat kan terug brengen.

Answer: Yes ☐ No ☒

Elaboration:

Dit aspect zit bij ons met name in de systeemlaag denk ik, zodat ik voor mijn specifieke deelgebied er niet zoveel mee te maken heb.

Q3) **Usability** – Is usability an aspect of quality that plays a role when *selecting* or *comparing* software patterns?
(De mate waarin een product of systeem gebruikt kan worden door gespecificeerde gebruikers om effectief, efficiënt en naar tevredenheid gespecificeerde doelen te bereiken in een gespecificeerde gebruikscontext.)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:

Klik hier als u tekst wilt invoeren.

When selecting or comparing software patterns does a software architect take into account if:

Herkenbaarheid van geschiktheid (Appropriateness recognisability)

De mate waarin gebruikers kunnen herkennen of een product of systeem geschikt is voor hun behoeften.

Answer: Yes ☐ No ☒
Elaboration:
Klik hier als u tekst wilt invoeren.

Leerbaarheid (Learnability)
De mate waarin een product of systeem gebruikt kan worden door gespecificeerde gebruikers om gespecificeerde leerdoelen te bereiken met betrekking tot het gebruik van het product of systeem met effectiviteit, efficiëntie, vrijheid van risico en voldoening, in een gespecificeerde gebruikscontext.

Answer: Yes ☐ No ☒
Elaboration:
Klik hier als u tekst wilt invoeren.

Bedienbaarheid (Operability)
De mate waarin een product of systeem attributen heeft die het makkelijk maken om het te bedienen en beheersen.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Voorkomen gebruikersfouten (User error protection)
De mate waarin het systeem gebruikers beschermt tegen het maken van fouten.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Volmaaktheid gebruikersinteractie (User interface aesthetics)

De mate waarin een gebruikersinterface het de gebruiker mogelijk maakt om een plezierige en voldoening gevende interactie te hebben.

Answer: Yes ☒ No ☐

Elaboration:

Ook hier wordt dit groot deel door systeemlaag geimplementeerd.

Toegankelijkheid (Accessibility)

De mate waarin een product of systeem gebruikt kan worden door mensen met de meest uiteenlopende eigenschappen en mogelijkheden om een gespecificeerd doel te bereiken in een gespecificeerde gebruikscontext

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Q4) **Prestatie-efficiëntie (Performance efficiency)** – Is efficiency an aspect of quality that plays a role when *selecting* or *comparing* software patterns?
(De prestaties in verhouding tot de hoeveelheid middelen gebruikt onder genoemde condities.)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:

Klik hier als u tekst wilt invoeren.

When selecting or comparing software patterns does a software architect take into account:

Snelheid (Time-behaviour)

De mate waarin antwoord- en verwerkingstijden en doorvoersnelheid van een product of systeem, tijdens de uitvoer van zijn functies, voldoet aan de wensen.

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Middelenbeslag (Resource utilization)

De mate waarin de hoeveelheid en type middelen die gebruikt worden door een product of systeem, tijdens de uitvoer van zijn functies, voldoet aan de wensen.

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Capaciteit (Capacity)

De mate waarin de maximale limieten van een product- of systeemparameter voldoet aan de wensen.

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Q5) **Onderhoudbaarheid (Maintainability)** – Is maintainability an aspect of quality that plays a role when *selecting* or *comparing* software patterns?
(De mate waarin een product of systeem effectief en efficiënt gewijzigd kan worden door de aangewezen beheerders.)

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Klik hier als u tekst wilt invoeren.

When selecting or comparing software patterns does a software architect take into account:

Modulariteit (Modularity)
De mate waarin een systeem of computerprogramma opgebouwd is in losstaande componenten zodat wijzigingen van een component minimale impact heeft op andere componenten.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Herbruikbaarheid (Reusability)
De mate waarin een bestaand onderdeel gebruikt kan worden in meer dan één systeem of bij het bouwen van een nieuw onderdeel.

Answer: Yes ☒ No ☐
Elaboration:
Klik hier als u tekst wilt invoeren.

Analyseerbaarheid (Analysability)

De mate waarin het mogelijk is om effectief en efficiënt de impact, van een geplande verandering van één of meer onderdelen, op een product of systeem te beoordelen, om afwijkingen en/of foutoorzaken van een product vast te stellen of om onderdelen te identificeren die gewijzigd moeten worden.

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Wijzigbaarheid (Modifiability)

De mate waarin een product of systeem effectief en efficiënt gewijzigd kan worden zonder fouten of kwaliteitsvermindering tot gevolg.

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Testbaarheid (Testability)

De mate waarin effectief en efficiënt testcriteria vastgesteld kunnen worden voor een systeem, product of component en waarin tests uitgevoerd kunnen worden om vast te stellen of aan die criteria is voldaan.

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Q6) **Overdraagbaarheid (Portability)** – Is portability an aspect of quality that plays a role when *selecting* or *comparing* software patterns?

(De mate waarin een systeem, product of component effectief en efficiënt overgezet kan worden van één hardware, software of andere operationele of gebruiksomgeving naar een andere.)

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:

Klik hier als u tekst wilt invoeren.

When selecting or comparing software patterns does a software architect take into account:

Aanpasbaarheid (Adaptability)

De mate waarin een product of systeem effectief en efficiënt aangepast kan worden voor andere of zich ontwikkelende hardware, software of andere operationele of gebruiksomgevingen.

Answer: Yes ☐ No ☒

Elaboration:

Dit is voor mij geen day to day business omdat de systeemlaag dit implementeert en dat we gebruik maken van bijv. jQuery.

Installeerbaarheid (Installability)

De mate waarin het product of het systeem effectief en efficiënt geïnstalleerd of verwijderd kan worden in een gespecificeerde omgeving.

Answer: Yes ☐ No ☒

Elaboration:

Klik hier als u tekst wilt invoeren.

Vervangbaarheid (Replaceability)

De mate waarin een product een ander specifiek softwareproduct, met hetzelfde doel in de zelfde omgeving, kan vervangen.

Answer: Yes ☐ No ☒
Elaboration:
Klik hier als u tekst wilt invoeren.

Q7) **Uitwisselbaarheid (Compatibility)**– Is compatibility an aspect of quality that plays a role when *selecting* or *comparing* software patterns?
(De mate waarin een product, systeem of component informatie uit kan wisselen met andere producten, systemen of componenten, en/of het de gewenste functies kan uitvoeren terwijl het dezelfde hard- of software-omgeving deelt.)

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Implementatie van zowel import/export via XML als CSV

When selecting or comparing software patterns does a software architect take into account:

Beïnvloedbaarheid (Co-existence)
De mate waarin een product zijn gewenste functies efficiënt kan uitvoeren terwijl het een gemeenschappelijke omgeving en middelen deelt met andere producten, zonder nadelige invloed op enig ander product.

Answer: Yes ☒ No ☐

Elaboration:

Ja en Nee. Omdat we een Multi tentant product zijn betekent dat dat op die ene omgeving/database meerdere klanten actief zijn. Het is dus zeer belangrijk voor ons dat de ene klant niet de data van de andere klant kan benaderen. Anderzijds moet ik misschien No kiezen omdat het eigenlijk geen ander product is.

Koppelbaarheid (Interoperability)

De mate waarin twee of meer systemen, producten of componenten informatie kunnen uitwisselen en de uitgewisselde informatie kunnen gebruiken.

Answer: Yes ☒ No ☐

Elaboration:

Denk aan uitwisseling tussen ons product en web shop

Q8) **Beveiligbaarheid (Security) -** Is security an aspect of quality that plays a role when *selecting* or *comparing* software patterns?
(De mate waarin een product of systeem informatie en gegevens beschermt zodat personen, andere producten of systemen de juiste mate van gegevenstoegang hebben passend bij hun soort en niveau van autorisatie.

Answer: Yes ☒ No ☐

(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:

Uiteraard heeft een web product te maken met security.

When selecting or comparing software patterns does a software architect take into account:

Vertrouwelijkheid (Confidentiality)

De mate waarin een product of systeem er voor zorgt dat gegevens alleen toegankelijk zijn voor diegenen die geautoriseerd zijn.

Answer: Yes ☒ No ☐

Elaboration:

Ja enerzijds hoe de verschillende users van een klant afhankelijk van rechten sommige dingen wel of niet mogen zien. Anderzijds hoe wij als software leverancier omgaan met de toegang tot de data van de klant voor de eigen mensen.

Integriteit (Integrity)

De mate waarin een systeem, product of component ongeautoriseerde toegang tot of aanpassing van computerprogramma's of gegevens verhindert.

Answer: Yes ☒ No ☐

Elaboration:

Klik hier als u tekst wilt invoeren.

Onweerlegbaarheid (Non-repudiation)

De mate waarin kan worden bewezen dat acties of gebeurtenissen plaats hebben gevonden, zodat later deze acties of gebeurtenissen niet ontkend kunnen worden.

Answer: Yes ☒ No ☐

Elaboration:

Dit is meer vanuit de optiek, dat data op een bepaalde manier opslaan zodat bij analyse van problemen/data min of meer eenduidig kunnen concluderen wat er is gebeurd.

Verantwoording (Accountability)
De mate waarin acties van een entiteit getraceerd kunnen worden naar die specifieke entiteit.

Answer: Yes ☒ No ☐
Elaboration:
Vaak functioneel nodig. Komt een factuur nu vanuit een verkooporder of is deze aangemaakt bij de verkoop van een active of nog een andere bron. Is dat wat je bedoeld?

Authenticiteit (Authenticity)
De mate waarin bewezen kan worden dat de identiteit van een onderwerp of bron is zoals wordt beweerd. De mate waarin een claim over de oorsprong of de auteur van de informatie verifieerbaar is, bijvoorbeeld aan handschrift.

Answer: Yes ☐ No ☒
Elaboration:
Klik hier als u tekst wilt invoeren.

Q9) **Ease of learning** – Is ease of learning a pattern characteristic that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- How long does it take for a developer to master the pattern?
- Is the pattern easy to understand?

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Klik hier als u tekst wilt invoeren.

Q10) **Ease of implementation** – Is ease of implementation a pattern characteristic that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- How long it takes to implement the pattern?
- What difficulties can be expected when implementing the pattern?

Answer: Yes ☒ No ☐
(If the attribute can play a role or might play a role when selecting or comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Klik hier als u tekst wilt invoeren.

Q11) **Dependency** – Is dependency a pattern characteristic that plays a role when *selecting* or *comparing* software patterns?

When selecting or comparing software patterns does a software architect take into account if:

- Does the pattern need other patterns to function?
- Does the pattern belong to a group of patterns?

Answer: Yes ☐ No ☒
(If the attribute can play a role or might play a role when selecting or

comparing software patterns, “yes” should be answered. Only when an attribute is never used in these cases, “no” should be answered.)

Elaboration:
Geen voorbeeld van.

Research question

RQ2) How can attributes of software patterns be quantified in a manner that allows for comparison?

Introduction

In order for quality attributes to be comparable, they need to be quantified. If the same scale and quantification method is used, the results of the evaluation can be compared with one another. This is done by quantifying the results of an evaluation session where multiple experts give their scores. Evaluation of software architectures is common practice in the domain of software architecture. The characteristics of several popular assessment methods have been deduced and form the basis of the following questions. Based on these questions a quality attribute quantification method for software patterns is created.

Q1) **SAAM** – Is it important to give a value to quality attributes based on multiple scenarios?

It is possible that a quality attribute would be valued differently under different circumstances. For example the impact on performance of a software product by a software pattern can be very different for a large scale application then it is for a smaller project. Including these scenarios might give the software architect more detailed information on the software pattern.

Answer: Yes ☒ No ☐

Elaboration:
Dit speelt niet altijd een rol, maar het voorbeeld van performance spreekt wel aan. Soms moet je na verloop van tijd een stuk refactoren omdat het functioneel wel correct is, maar de gebruikte opzet voor teveel vertraging zorgt.

Q2) Should it be possible to give a **negative value** to a quality attribute?

The influence of a software pattern on the software product can be negative at times. Each pattern has strengths and weaknesses that affect quality attributes. It might be important for a software architect to know that the implementation of a certain software pattern has a negative influence on the quality of software.

Answer: Yes ☒ No ☐

Elaboration:

Q3) Should **trade-offs** between quality attributes be included?

In many cases there are certain trade-offs between quality attributes. A pattern which adds more security can do so at the cost of performance. Although all quality attributes are valued, the specific trade-offs could also be included in the evaluation. Adding information on trade-offs would give additional information on quality attributes and their relationships.

Answer: Yes ☒ No ☐

Elaboration:
Klik hier als u tekst wilt invoeren.

Q4) At what **scale** should attributes be measured?

The scale determines the level of detail at which quality attributes are valued. A larger scale can provide more detail, but only if the evaluation yields meaningful information that can be quantified on such scale. In other words, we want to provide the highest level of detail at which the evaluation can still provide meaningful results.

Answer: 3-point scale ☒ 5-point scale ☒ 10-point scale ☐ 100-point scale ☐

Elaboration:
Klik hier als u tekst wilt invoeren.

Q5) Should the **experience** of a software architect be **weighted**?

When an evaluation of software patterns is performed, it is done by a group of experts. The experience of an expert accounts for much of the knowledge on software patterns and their quality attributes. When the results of an evaluation session are quantified the experience of each software architect can be weighted in the score to enable a higher influence by those who have more experience.

Answer: Yes ☒ No ☐

Elaboration:
Klik hier als u tekst wilt invoeren.

Q6) Should the software architect have **experience using the pattern**?

When evaluating a pattern the knowledge of multiple experts is used as input. However, it might not be possible for an expert to perform a thorough evaluation based on general understanding of patterns without having experience with the pattern that is being evaluated. Can a pattern be evaluated based on a description of the pattern without having experience using it?

Answer: Yes ☒ No ☒

Elaboration:
I'm not sure

F. Focus group session 1 – Score table

Pattern name: Observer

Attribute name	Group score	Average	Standard deviation	Participant 1	Participant 2	Participant 3	Participant 4
Functional suitability	0	1,25	2,50	0	5	0	0
Functional completeness	0	1,25	2,50	0	5	0	0
Functional correctness	0	0,75	1,50	0	3	0	0
Functional appropriateness	0	1,25	2,50	0	5	0	0
Performance efficiency	-2	-0,75	1,89	-2	-1	2	-2
Time-behavior	-2	-0,50	1,73	-1	-1	2	-2
Resource utilization	-2	-0,50	1,73	-1	-1	2	-2
Capacity	-2	0,25	2,06	2	-1	2	-2
Compatibility	1	0,75	0,96	1	0	2	0
Co-existence	1	0,50	1,00	0	0	2	0
Interoperability	1	0,75	0,96	1	0	2	0
Usability	0	0,00	0,00	0	0	0	0
Appropriateness recognisability	0	0,00	0,00	0	0	0	0
Learnability	0	0,00	0,00	0	0	0	0
Operability	0	0,00	0,00	0	0	0	0
User error protection	0	0,00	0,00	0	0	0	0
User interface aesthetics	0	0,00	0,00	0	0	0	0
Accessibility	0	0,00	0,00	0	0	0	0

Reliability	1	1,50	1,91	4	2	0	0
Maturity	1	1,00	2,00	0	4	0	0
Availability	1	1,50	1,91	4	2	0	0
Fault tolerance	1	1,50	1,91	4	2	0	0
Recoverability	1	0,50	1,00	0	2	0	0
Security	0	-0,50	1,00	-2	0	0	0
Confidentiality	0	0,00	0,00	0	0	0	0
Integrity	0	-0,75	1,50	-3	0	0	0
Non-repudiation	0	0,00	0,00	0	0	0	0
Accountability	0	-0,75	1,50	-3	0	0	0
Authenticity	0	0,00	0,00	0	0	0	0
Maintainability	3	3,50	1,00	5	3	3	3
Modulatiry	4	4,25	0,96	5	4	3	5
Reusability	2	2,50	1,73	3	3	0	4
Analysability	2	0,00	3,65	-4	4	2	-2
Modifiability	4	4,00	0,82	5	4	4	3
Testability	3	2,75	2,63	5	-1	3	4
Portability	0	0,25	0,50	0	0	1	0
Adaptability	0	0,25	0,50	0	0	1	0
Installability	0	0,25	0,50	0	0	1	0
Replaceability	0	0,25	0,50	0	0	1	0
Ease of implementation	4	4,25	0,50	4	4	5	4
Ease of learning	3	2,25	2,50	2	-1	5	3

G. Focus group session 2 – Score table

Pattern name: Check point

Attribute name	Group score	Avg.	SD	P1	P2	P3	P4	P5	P6	P7	P8	P9
Performance efficiency	**-1**	**0,00**	**1,07**	-1	-1	-1	-1	1	1	1	1	-1
Time-behavior	-1	-1,00	0,00	-1	-1	-1	-1	-1	-1	-1	-1	-1
Resource utilization	-1	-1,00	0,00	-1	-1	-1	-1	-1	-1	-1	-1	-1
Capacity	-1	-0,88	0,35	-1	0	-1	-1	-1	-1	-1	-1	-1
Compatibility	**0**	**0,38**	1,06	0	0	-1	2	0	0	2	0	0
Co-existence	0	0,00	0,00	0	0	0	0	0	0	0	0	0
Interoperability	0	0,00	0,00	0	0	0	0	0	0	0	0	0
Usability	**0**	**0,13**	**0,83**	0	-1	0	1	-1	1	1	0	1
Appropriateness recognisability	0	0,13	0,83	0	-1	0	1	-1	1	1	0	1
Learnability	0	0,13	0,83	0	-1	0	1	-1	1	1	0	1
Operability	0	0,13	0,83	0	-1	0	1	-1	1	1	0	1
User error protection	0	0,13	0,83	0	-1	0	1	-1	1	1	0	1
User interface aesthetics	0	0,13	0,83	0	-1	0	1	-1	1	1	0	1
Accessibility	0	0,13	0,83	0	-1	0	1	-1	1	1	0	1
Reliability	**1**	**0,63**	**0,52**	0	1	1	0	1	1	0	1	1
Maturity	1	0,63	0,52	0	1	1	0	1	1	0	1	1
Availability	1	0,63	0,52	0	1	1	0	1	1	0	1	1
Fault tolerance	1	0,63	0,52	0	1	1	0	1	1	0	1	1

Recoverability	1	0,63	0,52	0	1	1	0	1	1	0	1	1
Security	**2**	**2,50**	**0,53**	3	3	2	3	2	2	2	3	2
Confidentiality	2	2,50	0,53	3	3	2	3	2	2	2	3	2
Integrity	2	2,50	0,53	3	3	2	3	2	2	2	3	2
Non-repudiation	2	2,50	0,53	3	3	2	3	2	2	2	3	2
Accountability	2	2,50	0,53	3	3	2	3	2	2	2	3	2
Authenticity	2	2,50	0,53	3	3	2	3	2	2	2	3	2
Maintainability		"	"									
Modulatiry		"	"									
Reusability		"	"									
Analysability		"	"									
Modifiability		"	"									
Testability		"	"									
Portability		"	"									
Adaptability		"	"									
Installability		"	"									
Replaceability		"	"									
Ease of implementation		"	"									
Ease of learning		"	"									

H. Focus group session 3 – Score table

Pattern name: Check point

Attribute name	Group score	Avg.	SD	P1	P2	P3	P4	P5	P6	P7	P8
Performance efficiency	-1	-1,00	0,76	-1	0	-2	-1	0	-1	-1	-2
Time-behavior		''	''								
Resource utilization		''	''								
Capacity		''	''								
Compatibility	0	0,88	1,25	1	1	2	3	0	1	-1	0
Co-existence		''	''								
Interoperability		''	''								
Usability	0	0,00	1,07	0	1	0	0	-1	2	-1	-1
Appropriateness recognisability		''	''								
Learnability		''	''								
Operability		''	''								
User error protection		''	''								
User interface aesthetics		''	''								
Accessibility		''	''								
Reliability		-0,25	1,04	0	-1	-1	-1	0	-1	2	0
Maturity		''	''								
Availability		''	''								
Fault tolerance		''	''								
Recoverability		''	''								

Security	2	1,88	**1,25**	0	0	3	2	2	3	2	3
Confidentiality		''	''								
Integrity		''	''								
Non-repudiation		''	''								
Accountability		''	''								
Authenticity		''	''								
Maintainability	0	0,38	**0,74**	0	1	0	1	0	1	-1	1
Modulatiry		''	''								
Reusability		''	''								
Analysability		''	''								
Modifiability		''	''								
Testability		''	''								
Portability	1	1,00	0,93	1	2	2	2	1	0	0	0
Adaptability		''	''								
Installability		''	''								
Replaceability		''	''								
Ease of implementation		0,38	**1,85**	-1	0	-2	2	2	2	-2	2
Ease of learning	2	1,88	**0,83**	1	2	3	3	1	2	1	2