An Analytic Evaluation of the SaCS Pattern Language – Including Explanations of Major Design Choices

André Alexandersen Hauge
Institute for energy technology, Halden, Norway
University of Oslo, Norway
andre.hauge@hrp.no

Ketil Stølen
SINTEF ICT, Oslo, Norway
University of Oslo, Norway
ketil.stolen@sintef.no

***Abstract*—In this paper, we present an analytic evaluation of the Safe Control Systems (SaCS) pattern language for the development of conceptual safety designs. By a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The SaCS pattern language may express basic patterns on different aspects of relevance for conceptual safety designs. SaCS may also be used to combine basic patterns into composite patterns. A composite pattern may be instantiated into a conceptual safety design. A framework for evaluating modelling languages is used to conduct the evaluation. The quality of a language is within the framework expressed by six appropriateness factors. A set of requirements is associated with each appropriateness factor. The extent to which these requirements are fulfilled are used to judge the quality. We discuss the fulfilment of the requirements formulated for the SaCS language on the basis of the theoretical, technical, and practical considerations that were taken into account and shaped the SaCS language.**

Keywords–pattern language, analytic evaluation, design conceptualisation, safety.

I. Introduction

A pattern describes a particular recurring problem that arises in a specific context and presents a well-proven generic scheme for its solution [1]. A pattern language is a language for specifying patterns making use of patterns from a vocabulary of existing patterns and defined rules for combining these [2]. A safety critical system [3] is a system *"whose failure could result in loss of life, significant property damage, or damage to the environment"*. With a conceptual safety design we mean an early stage specification of system requirements, system design, and safety case for a safety critical system. The Safe Control Systems (SaCS) pattern language has been designed to facilitate the specification of patterns to support the development of conceptual safety designs. The intended users of SaCS are system engineers, safety engineers, hardware and software engineers.

This paper conducts an analytic evaluation of the suitability of the SaCS pattern language for its intended task. A framework for analysing languages known as the semiotic quality framework (SEQUAL) [4] is used as a basis for the evaluation. The appropriateness of a language for its intended task is in the framework characterised by six appropriateness factors [4]: domain, modeller, participant, comprehensibility, tool, and organisational. A set of requirements is presented for each appropriateness factor in order to characterise more precisely what is expected from our language in order to be appropriate. The requirements represent the criteria for judging what is appropriate of a language for conceptual safety design, independent of SaCS being appropriate or not. We motivate our choices and discuss to what extent the requirements are fulfilled.

The remainder of this article is structured as follows: Section II provides a short introduction to the SaCS pattern language. Section III discusses evaluation approaches and motivates the selection of SEQUAL. Section IV motivates the selection of requirements and conducts an evaluation of the SaCS language with respect to these requirements for each appropriateness factor. Section V presents related work on pattern-based development. Section VI draws the conclusions.

II. Background on The SaCS Pattern Language

Fig. 1 defines a composite pattern according to the syntax of SaCS [5]. The composite described in Fig. 1 is named *Safety Requirements* and consists of the basic patterns *Hazard Analysis*, *Risk Analysis*, and *Establish System Safety Requirements*. The basic patterns are specified separately in a structured manner comparable to what can be found in the literature [1][2][6][7][8][9][10][11] on patterns.

In Fig. 1, the horizontal line separates the declaration part of the composite pattern from its content. The icon placed below the identifier *Safety Requirements* signals that this is a composite pattern. Every pattern in SaCS is parametrised. An input parameter represents the information expected to be provided when applying a pattern in a context. An output parameter represents the expected outcome of applying a pattern in a given context. The inputs to *Safety Requirements* are listed inside square brackets to the left of the icon, i.e., *ToA* and *Haz*. The arrow pointing towards the brackets symbolises input. The output of the pattern is also listed inside square brackets, but on the right-hand side of the icon, i.e., *Req*. The arrow pointing away from the brackets symbolises output. An icon placed adjacent to a parameter identifier denotes its type. The parameters *ToA*, *Haz*, *HzLg*, and *Risks* in Fig. 1 are of type *documentation*, while *Req* is of type *requirement*. The inputs and outputs of a composite are always publicly accessible.

 ISBN: 978-1-61208-343-8

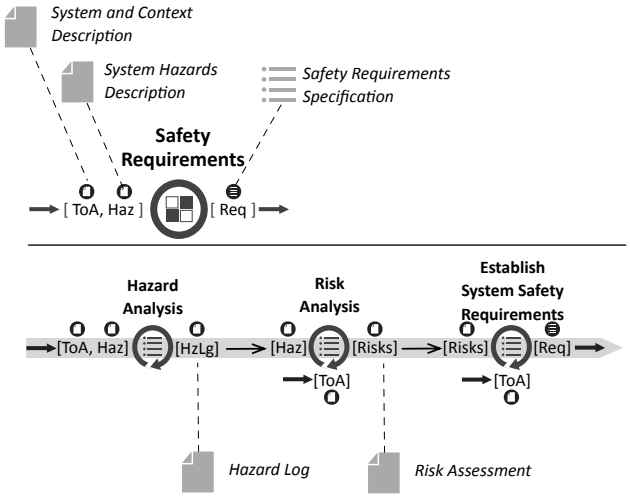

Figure 1. A composite pattern named Safety Requirements

A particular instantiation of a parameter is documented by a relation that connects a parameter with its associated development artefact. In Fig. 1, a grey icon placed adjacent to an identifier of a development artefact classifies what kind of artefact that is referenced. A dotted drawn line connecting a parameter with an artefact represents an *instantiates* relation. Instantiations of parameters expressed in Fig. 1 are:

- the document artefact *System and Context Description* instantiates *ToA*.
- the document artefact *System Hazards Description* instantiates *Haz*.
- the requirement artefact *Safety Requirements Specification* instantiates *Req*.
- the document artefact *Hazard Log* instantiates *HzLg*.
- the document artefact *Risk Assessment* instantiates *Risks*.

A one-to-many relationship exists between inputs in the declaration part of a composite and similarly named inputs with public accessibility (those pointed at by fat arrows) in the content part. The relationship is such that when *ToA* of *Safety Requirements* is instantiated (i.e., given its value by the defined relation to *System and Context Description*) then every correspondingly named input parameter contained in the composite is also similarly instantiated. A one-to-one relationship exists between an output parameter in the declaration part of a composite and a correspondingly named output parameter with public accessibility (those followed by a fat arrow) in the content part. The relationship is such that when *Req* of *Establish System Safety Requirements* is produced then *Req* of *Safety Requirements* is similarly produced.

The arrows (thin arrows) connecting basic patterns in the content part of *Safety Requirements* represent two instances of an operator known as the *assigns* relation. The *assigns* relations within *Safety Requirements* express that:

- The output *HzLg* of the pattern *Hazard Analysis* is assigned to the input *Haz* of the pattern *Risk Analysis*.
- The output *Risks* of the pattern *Risk Analysis* is assigned to the input *Risks* of the pattern *Establish System Safety Requirements*.

That the three basic patterns are process patterns follows from the icon below their respective identifiers. There are six different kinds of basic patterns in SaCS, each represented by a specific icon.

III. Evaluation framework

Mendling et al. [12] describe two dominant approaches in the literature for evaluating the quality of modelling approaches: (1) top-down quality frameworks; (2) bottom-up metrics that relate to quality aspects. The most prominent top-down quality framework according to [12] is SEQUAL [4][13][14]. The framework is based on semiotic theory (the theory of signs) and is developed for evaluating the quality of conceptual models and languages of all kinds. Moody et al. [15] report on an empiric study involving 194 participants on the use of SEQUAL and concludes that the study provides strong support for the validity of the framework. Becker et al. [16] present a guideline-based approach as an alternative to SEQUAL. It addresses the six factors: correctness, clarity, relevance, comparability, economic efficiency, and systematic design. Mendling et al. [12] also discuss a number of bottom-up metrics approaches. Several of these contributions are theoretic without empirical validation according to the authors.

We have chosen to apply the SEQUAL framework for our evaluation as it is a general framework applicable to different kinds of languages [4] whose usefulness has been confirmed in experiments [15]. Furthermore, an analytic evaluation is preferred over a metric-based approach due to project limitations. An analytic evaluation is also a suitable complement to the experience-based evaluations of SaCS presented in [17][18].

The appropriateness of a modelling language for a specific task is in SEQUAL related to the definition of the following sets: the set of goals G for the modelling task; its domain D in the form of the set of all statements that can be stated about the situation at hand; the relevant knowledge of the modeller Km and other participants Ks involved in the modelling task; what persons involved interpret the models to say I; the language L in the form of the set of all statements that can be expressed in the language; relevant tool interpretation T of the models; and what is expressed in the models M.

Fig. 2 is adopted from [13] and illustrates the relationships between the different sets in SEQUAL. The quality of a language L is expressed by six appropriateness factors. The quality of a model M is expressed by nine quality aspects.

In the following, we will not address the different quality aspects of a model M but rather address the quality of the SaCS pattern language.

The appropriateness factors indicated in Fig. 2 are related to different properties of the language under evaluation. The appropriateness factors are [4]:

- *Domain appropriateness*: the language should be able to represent all concepts in the domain.

 ISBN: 978-1-61208-343-8

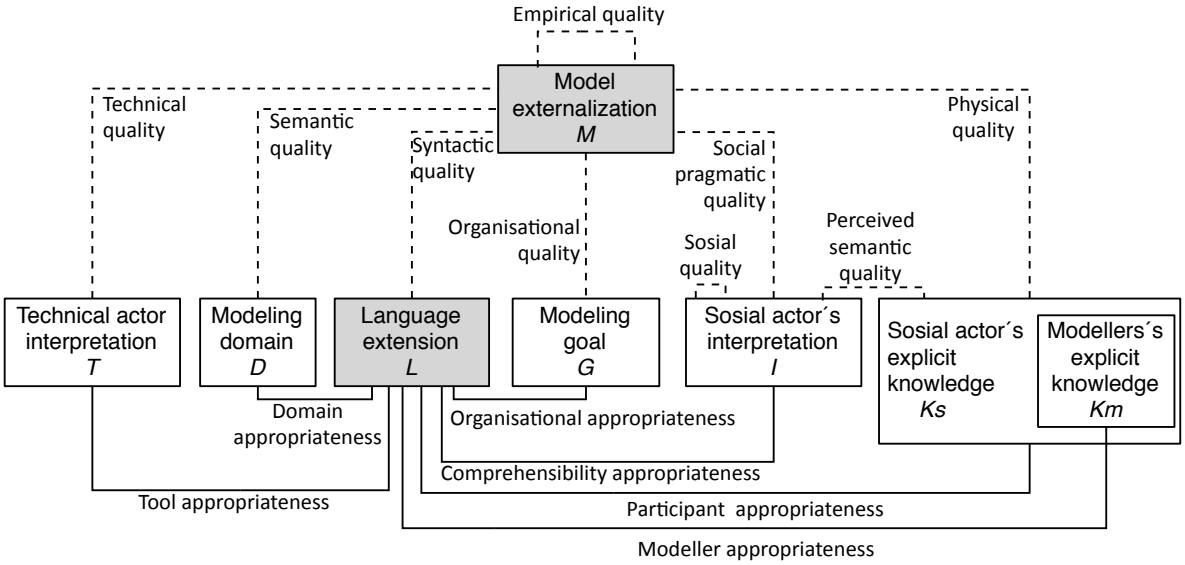

Figure 2. The quality framework (adopted from [19])

- *Modeller appropriateness*: there should be no statements in the explicit knowledge of the modeller that cannot be expressed in the language.
- *Participant appropriateness*: the conceptual basis should correspond as much as possible to the way individuals who partake in modelling perceive reality.
- *Comprehensibility appropriateness*: participants in the modelling should be able to understand all the possible statements of the language.
- *Tool appropriateness*: the language should have a syntax and semantics that a computerised tool can understand.
- *Organisational appropriateness*: the language should be usable within the organisation it targets such that it fits with the work processes and the modelling required to be performed.

A set of requirements is associated with each appropriateness factor. The extent to which the requirements are fulfilled are used to judge the quality of the SaCS pattern language for its intended task. The requirements are defined on the basis of requirements found in the literature on SEQUAL [4].

IV. THE EVALUATION

A necessary step in the application of SEQUAL [4][13] is to adapt the evaluation to account for the modelling needs. This amounts to expressing what the different appropriateness factors of the framework represent in the particular context of the evaluation in question. In particular, the modelling needs are detailed by the definition of a set of criteria for each of the appropriateness factors.

Table I introduces the criteria for evaluating the suitability of the SaCS pattern language for its intended task. In the first column of Table I, the two letters of each requirement identifier identify the appropriateness factor addressed by the requirement, e.g., DA for Domain Appropriateness.

TABLE I. OVERVIEW OF EVALUATION CRITERIA

ID	Requirement
DA.1	The language must include the concepts representing best practices within conceptual safety design.
DA.2	The language must support the application of best practices within conceptual safety design.
MA.1	The language must facilitate tacit knowledge externalisation within conceptual safety design.
MA.2	The language must support the modelling needs within conceptual safety design.
PA.1	The terms used for concepts in the language must be the same terms used within safety engineering.
PA.2	The symbols used to illustrate the meaning of concepts in the language must reflect these meanings.
PA.3	The language must be understandable for people familiar with safety engineering without specific training.
CA.1	The concepts and symbols of the language should differ to the extent they are different.
CA.2	It must be possible to group related statements in the language in a natural manner.
CA.3	It must be possible to reduce model complexity with the language.
CA.4	The symbols of the language should be as simple as possible with appropriate use of colour and emphasis.
TA.1	The language must have a precise syntax.
TA.2	The language must have a precise semantics.
OA.1	The language must be able to express the desired conceptual safety design when applied in a safety context.
OA.2	The language must ease the comprehensibility of best practices within conceptual safety design for relevant target groups like system engineers, safety engineers, hardware and software engineers.
OA.3	The language must be usable without the need of costly tools.

The different appropriateness factors are addressed successively in Section IV-A to Section IV-F according to the order in Table I. Each requirement from Table I is discussed. A requirement identifier is presented in a **bold** font when first introduced in the text followed by the associated requirement and an evaluation of the extent to which the requirement is fulfilled by SaCS.

A. Domain appropriateness

DA.1 The language must include the concepts representing best practices within conceptual safety design.

In the SaCS language, there are currently 26 basic patterns [17][18] on different concepts within conceptual safety design.

Each pattern may be referenced by its unique name. Three of the currently available basic patterns are referenced in Fig. 1 and are named *Hazard Analysis*, *Risk Analysis* and *Establish System Safety Requirements*.

Fig. 3 presents the icons used for basic SaCS patterns and indicates a categorisation. The three icons to the left are used for categorising patterns providing development guidance with a strong processual focus. The three icons to the right are used for categorising patterns providing development guidance with a strong product focus. Different kinds of patterns express different concepts and best practices within development of safety critical systems. The combined use of patterns from different categories facilitates development of conceptual safety designs.

Figure 3. Icons for the different kinds of basic pattern references

Habli and Kelly [20] describe the two dominant approaches in safety standards for providing assurance of safety objectives being met. These are: (1) the process-based approach; (2) the product-based approach. Within the process-based approach, safety assurance is achieved on the basis of evidence from the application of recommended or mandatory development practices in the development life cycle. Within the product-based approach, safety assurance is achieved on the basis of product specific evidences that meet safety requirements derived from hazard analysis. The practice within safety standards as described above motivate our categorisation into the process assurance and the product assurance pattern groups.

The safety property of a system is addressed on the basis of a demonstration of the fulfilment of safety objectives. Seven nuclear regulators [21] define a safety demonstration as *"a set of arguments and evidence elements that support a selected set of dependability claims - in particular the safety - of the operation of a system important to safety used in a given plant environment"*. Although it is the end system that is put into operation, evidences supporting safety claims are produced throughout the system life cycle and need to be systematically gathered from the very beginning of a development project [21]. The safety case approach represents a means for explicitly presenting the structure of claims, arguments, and evidences in a manner that facilitates evaluation of the rationale and basis for claiming that safety objectives are met. The safety case approach is supported by several authors [10][20][21][22]. What is described above motivates the need for patterns supporting safety case specification in addition to patterns on requirements elicitation and system design specification.

As indicated above, in the design of the SaCS pattern language we have as much as possible selected keywords and icons in the spirit of leading literature within the area. This indicates that we at least are able to represent a significant part of the concepts of relevance for conceptual safety design.

DA.2 The language must support the application of best practices within conceptual safety design.

Safety standards [23] may demand a number of activities to be performed in which certain activities must be applied in a specific sequence. Safety standards [23] may also describe the expected inputs and outputs of different activities and in this sense state what is the expected content of deliverables that allows a transition from one activity to the next. According to Krogstie [4], the main phenomena in languages that accommodate a behavioural modelling perspective are states and transitions between states. In this sense, the language should support the modelling of the application of best practices according to a behavioural modelling perspective.

Fig. 4 presents the icons for the different kinds of parameters and artefact references in SaCS. The *documentation parameter* and the *documentation artefact reference* types (represented visually by the icons presented in Fig. 4) are defined in order to allow a generic classification of parameters and artefacts that may not be classified as requirement, design, or safety case. An example may be the result of risk analysis that is an intermediate result in conceptual safety design and an input to an activity on the specification of safety requirements [23][24]. The process of deriving safety requirements on the basis of an assessment of hazards is expressed by a chain of patterns as presented in Fig. 1. The outcome of applying the last pattern in the chain is a requirements specification. The last pattern cannot be applied before the required inputs are produced.

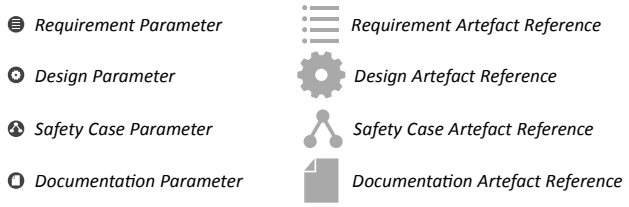

Figure 4. Icons for the different kinds of parameters and artefact references

Fig. 5 presents the symbolic representation of the different relations in SaCS. Relations define transitions between patterns or dependencies between elements within a composite pattern definition. The reports [17][18] define the concepts behind the different relations and exemplify the practical use of all the concepts in different scenarios. Fig. 1 is explained in Section II and exemplify a composite pattern containing five instances of the *instantiates* relation and two instances of the *assigns* relation.

Figure 5. Symbols for the different kinds of relations

The need for the different relations presented in Fig. 5 is motivated by the practices described in different standards and guidelines, e.g., IEC 61508 [23], where activities like hazard identification and hazard analysis are required to be performed sequentially and where the output of one activity is assigned as input to another activity. Thus, we need a concept of assignment. In SaCS, this is defined by an *assigns*

 ISBN: 978-1-61208-343-8

relation between patterns. When performing an activity like hazard analysis, the results from the application of a number of methods may be combined and used as input. Two widely known methods captured in two different basic SaCS patterns are Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA). A concept for combining results is needed in order to model that the results from applying several patterns as FMEA and FTA are combined into a union consisting of every individual result. In SaCS, this is defined by a *combines* relation between patterns. A *details* relation is used to express that the result of applying one pattern is further detailed by the application of a second pattern. Functional safety is an important concept in IEC 61508 [23]. Functional safety is a part of the overall safety that depends on a system or equipment operating correctly in response to its inputs. Furthermore, functional safety is achieved when every specified safety function is carried out and the level of performance required of each safety function is met. A *satisfies* relation between a pattern for requirements elicitation and a pattern for system design expresses that the derived system satisfies the derived requirements. Safety case patterns supports documenting the safety argument. A *demonstrates* relation between a safety case pattern and a design pattern expresses that the derived safety argument represents a safety demonstration for the derived system.

Fig. 6 and Fig. 7 illustrate how the intended instantiation order of patterns may be visualised. The direction of the arrow indicates the pattern instantiation order; patterns (or more precisely the patterns referred to graphically) placed closer to the starting point of the arrow are instantiated prior to patterns placed close to the tip of the arrow. Patterns may be instantiated in parallel and thus have no specific order; this is visualised by placing pattern references on separate arrows.

Figure 6. Serial instantiation

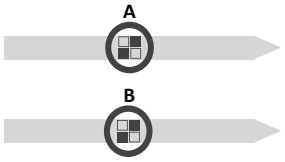

Figure 7. Parallel instantiation

As argued above, the SaCS language facilitates the application of best practices within safety design and mirrors leading international standards within the area; in particular IEC 61508. We therefore think it is fair to say that the language to a large extent fulfils DA.2.

B. Modeller appropriateness

MA.1 The language must facilitate tacit knowledge externalisation within conceptual safety design.

As already mentioned, the current version of the language contains 26 basic patterns. The basic patterns are documented in [17] and [18]. The patterns are defined on the basis of safety engineering best practices as defined in international standards and guidelines [21][23][24][25][26][27] and other sources on safety engineering. The limited number of basic patterns currently available delimit what can be modelled in a composite pattern. Defining more basic patterns will provide a better coverage of the tacit knowledge that can be externalised. A user may easily extend the language. A basic pattern, e.g., the pattern *Hazard Analysis* [17] referenced in Fig. 1, is defined in a simple structure of named sections containing text and illustrations according to a common format. The format is thoroughly detailed in [5].

Table II compares the overall format of basic SaCS patterns to pattern formats in the literature. We have chosen a format that resembles that of Alexander et al. [2] with the addition of the sections “Pattern signature”, “Intent”, “Applicability”, and “Instantiation rule”. The signature, intent, and applicability sections of basic patterns are documented in such a manner that the context section provided in [2] is not needed. The format in [2] is a suitable basis as it is simple, well-known, and generally applicable for specifying patterns of different kinds. The format provided by Gamma et al. [8] is also simple and well-known, but tailored specifically for capturing patterns for software design.

All in all, we admit that there may be relevant tacit knowledge that is not easily externalised as the SaCS language is today. However, the opportunity of increasing the number of basic patterns makes it possible to at least reduce the gap.

MA.2 The language must support the modelling needs within conceptual safety design.

IEC 61508 [23] is defined to be applicable across all industrial domains developing safety-related systems. As already mentioned, a key concept within IEC 61508 is functional safety. Functional safety is achieved according to [23] by adopting a broad range of principles, techniques and measures.

A key concept within SaCS is that principles, techniques, methods, activities, and technical solutions of different kinds are defined within the format of basic patterns. A limited number of concerns are addressed by each basic pattern. A

TABLE II. PATTERN FORMATS IN THE LITERATURE COMPARED TO BASIC SACS PATTERNS [5]

	[6]	[2]	[8]	[9]	[10]	[28]	[29]	[30]	[5]
Name	✓	✓	✓	✓	✓	✓	✓	✓	✓
Also known as			✓		✓				
Pattern signature									✓
Intent			✓		✓				✓
Motivation			✓		✓				
Applicability			✓		✓				✓
Purpose							✓		
Context	✓	✓				✓	✓		
Problem	✓	✓		✓		✓		✓	✓
Forces	✓					✓			
Solution	✓	✓		✓		✓	✓	✓	✓
Structure			✓		✓				
Participants			✓		✓				
Collaborations			✓		✓				
Consequences			✓		✓				
Implementation			✓		✓				
Sample code			✓						
Example					✓				
Compare							✓		
Instantiation rule									✓
Related patterns		✓	✓	✓	✓	✓			✓
Known uses	✓	✓		✓		✓			✓

 ISBN: 978-1-61208-343-8

specific combination of patterns is defined within a composite pattern. A composite pattern is intended to address the overall challenges that appear in a given development context. Individual patterns within a composite only address a subset of the challenges that need to be solved in the context. A composite may be defined prior to work initiation in order to define a plan for the application of patterns. Another use may be to refine a composite throughout the work process. This is exemplified in [17] and [18]. A composite may also be defined once patterns have been applied in order to document the work process. A composite representing a plan may be easily reused for documentation purposes by adding information on the instantiation of parameters.

C. Participants appropriateness

PA.1 The terms used for concepts in the language must be the same terms used within safety engineering.

Activities such as *hazard identification* and *hazard analysis* [26], methods such as *fault tree analysis* [31] and *failure mode effects analysis* [32], system design solutions including *redundant modules* and voting mechanisms [33], and practices like arguing safety on the basis of arguing that safety requirements are satisfied [21], are all well known safety engineering practices that may be found in different standards and guidelines [23][24][27]. The different concepts mentioned above are all reflected in basic SaCS patterns. Moreover, as already pointed out, keywords such as process assurance, product assurance, requirement, solution, safety case, etc. have all been selected based on leading terminology within safety engineering.

PA.2 The symbols used to illustrate the meaning of concepts in the language must reflect these meanings.

One commonly cited and influential article within psychology is that of Miller [34], on the limit of human capacity to process information. The limit, according to Miller, is seven plus or minus two elements. When the number of elements increases past seven, the mind may be confused in correctly interpreting the information. Thus, the number of symbols should be kept low in order to facilitate effective human information processing.

Lidwell et al. [35] describe iconic representation as *"the use of pictorial images to make actions, objects, and concepts in a display easier to find, recognize, learn, and remember"*. The authors describe four forms for representation of information with icons: similar, example, symbolic, and arbitrary. We have primarily applied the symbolic form to identify a concept at a higher level of abstraction than what may be achieved with the similar and example forms. We have also tried to avoid the arbitrary form where there is little or no relationship between a concept and its associated icon. Fig. 3, Fig. 4, Fig. 5, and Fig. 6 present the main icons in SaCS. In order to allow a flexible use of icons and keep the number of icons low, we have chosen to not define a dedicated icon for each concept but rather define icons that categorises several related concepts. A relatively small number of icons was designed in a uniform manner in order to capture intuitive representations of related concepts. As an example, the referenced basic patterns in Fig. 1 have the same icons linking them by category, but unique identifiers separating them by name.

PA.3 The language must be understandable for people familiar with safety engineering without specific training.

The SaCS language is simple in the sense that a small set of icons and symbols are used for modelling the application of patterns, basically: pattern references as in Fig. 3, parameters and artefact references as in Fig. 4, relations as in Fig. 5, and instantiation order as in Fig. 6. Guidance to the understanding of the language is provided in [5], where the syntax and the semantics of SaCS patterns are described in detail. The SaCS language comes with a structured semantics [5] that offers a schematic mapping from syntactical elements into text in English. Guidance to the application of SaCS is provided by the examples detailed in [17][18]. Although we have not tested SaCS on people unfamiliar with the language, we expect that users familiar with safety engineering may comprehend the concepts and the modelling on the basis of [5][17][18] within 2-3 working days.

D. Comprehensibility appropriateness

CA.1 The concepts and symbols of the language should differ to the extent they are different.

The purpose of the graphical notation is to represent a structure of patterns in a manner that is intuitive, comprehensible, and that allows efficient visual perception. The key activities performed by a reader in order to draw conclusion from a diagram are according to Larkin and Simon [36]: searching and recognising relevant information.

Lidwell et al. [35] present 125 patterns of good design based on theory and empirical research on visualisation. The patterns describe principles of designing visual information for effective human perception. The patterns are defined on the basis of extensive research on human cognitive processes. Some of the patterns are commonly known as Gestalt principles of perception. Ellis [37] provides an extensive overview of the Gestalt principles of perception building upon classic work from Wertheimer [38] and others. Gestalt principles capture the tendency of the human mind to naturally perceive whole objects on the basis of object groups and parts.

One of several Gestalt principles applied in the SaCS language is the principle of similarity. According to Lidwell et al. [35], the principle of similarity is such that similar elements are perceived to be more related than elements that are dissimilar.

The use of the similarity principle is illustrated by the composite pattern in Fig. 1. Although each referenced pattern has a unique name, their identical icons indicate relatedness. Different kinds of patterns are symbolised by the icons in Fig. 3. The icons are of the same size with some aspects of similarity and some aspects of dissimilarity such that a degree of relatedness may be perceived. An icon for pattern reference is different in shape and shading compared to an icon used for artefact reference (see Fig. 3 and Fig. 4). Thus, an artefact and a pattern should be perceived as representing quite different concepts.

CA.2 It must be possible to group related statements in the language in a natural manner.

There are five ways to organise information according to Lidwell et al. [35]: category, time, location, alphabet, and

 ISBN: 978-1-61208-343-8

continuum. The *category* refers to the organisation of elements by similarity and relatedness. An example of the application of the principle of categorisation [35] in SaCS is seen in the possibility to reduce the number of relations drawn between patterns when these are similar. Patterns in SaCS may have multiple inputs and multiple outputs as indicated in Fig. 1. Relations between patterns operate on the parameters. The brackets [ ] placed adjacent to a pattern reference denotes an ordered list of parameters. In order to avoid drawing multiple relations between two patterns, relations operate on the ordered parameter lists of the patterns by list-matching of parameters.

Fig. 8 exemplifies two different ways for expressing visually the same relationships between the composite patterns named *A* and *B*. The list-matching mechanism is used to reduce the number of relation symbols drawn between patterns to one, even though the phenomena modelled represents multiple similar relations. This reduces the visual complexity and preserves the semantics of the relationships modelled.

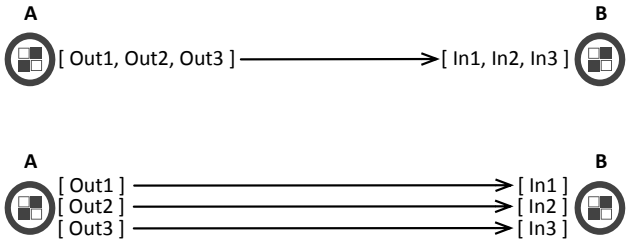

Figure 8. Alternative ways for visualising multiple similar relations

CA.3 It must be possible to reduce model complexity with the language.

Hierarchical organisation is the simplest structure for visualising and understanding complexity according to Lidwell et al. [35]. The SaCS language allows concepts to be organised hierarchically by specifying that one pattern is detailed by another or by defining composite patterns that reference other composite patterns in the content part.

Fig. 9 presents a composite pattern named *Requirements* that reference other composites as part of its definition. The contained pattern *Safety Requirements* is defined in Fig. 1. The contained pattern *Functional Requirements* is not defined and is referenced within Fig. 9 for illustration purposes. *Requirements* may be easily extended by defining composites supporting the elicitation of, e.g., performance requirements and security requirements, and later model the use of such patterns in Fig. 9. In Fig. 9, the output of applying the *Requirements* pattern is represented by the parameter *ReqSpec*. The *ReqSpec* parameter represents the result of applying the *combines* relation on the output *Req* of the composite *Safety Requirements* and the output *Req* of the composite *Functional Requirements*.

CA.4 The symbols of the language should be as simple as possible with appropriate use of colour and emphasis.

A general principle within visualisation according to Lidwell et al. [35] is to use colour with care as it may lead to misconceptions if used inappropriately. The authors points out that there is no universal symbolism for different colours.

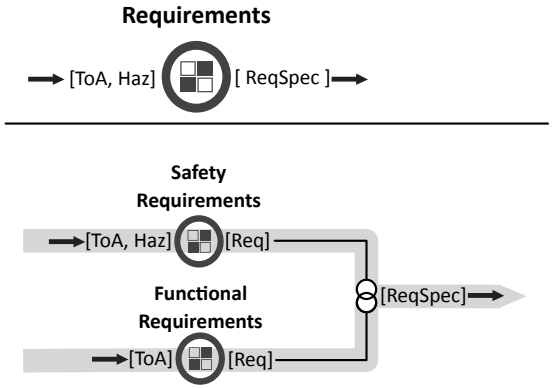

Figure 9. Composition of composites

As colour blindness is common the SaCS language applies different shades of grey in visualisations.

Fig. 10 illustrates how the SaCS language makes use of the three Gestalt principles of perception [35][39][38] known as: Figure-Ground; Proximity; and Uniform Connectedness. The Gestalt principles express mechanisms for efficient human perception from groups of visual objects.

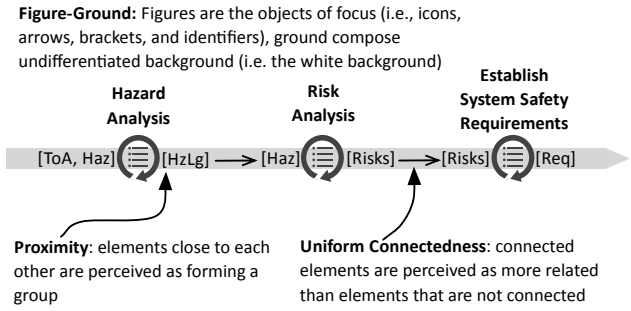

Figure 10. A fragment of Fig. 1 illustrating the use of Gestalt principles

E. Tool appropriateness

TA.1 The language must have a precise syntax.

The syntax of the SaCS language (see [5]) is defined in the EBNF [40] notation. EBNF is a meta-syntax that is widely used for describing context-free grammars.

TA.2 The language must have a precise semantics.

A structured semantics for SaCS patterns is defined in [5] in the form of a schematic mapping from pattern definitions, via its textual syntax in EBNF [40], to English. The non-formal representation of the semantics supports human interpretation rather than tools, although the translation procedure as described in [5] may be automated. The presentation of the semantics of patterns as a text in English was chosen in order to aid communication between users, possibly with different technical background, on how to interpret patterns.

F. Organisational appropriateness

OA.1 The language must be able to express the desired conceptual safety design when applied in a safety context.

 ISBN: 978-1-61208-343-8

The application of the SaCS pattern language produces composite patterns that are instantiated into conceptual safety designs. A composite pattern expresses a combination of basic patterns. The basic patterns express safety engineering best practices and concepts inspired by international safety standards and guidelines, e.g., [23][24][27]. International safety standards and guidelines describe concepts and practices for development of safety critical systems that may be perceived as commonly accepted. The SaCS pattern language is tested out in two cases. The first concerned the conceptualisation of a nuclear power plant control system, while the second addressed the conceptualisation of a railway interlocking system, fully detailed in [17] and [18], respectively. In both cases it was possible to derive a conceptual safety design using the SaCS language as support as well as model how patterns were applied as support.

OA.2 The language must ease the comprehensibility of best practices within conceptual safety design for relevant target groups like system engineers, safety engineers, hardware and software engineers.

We have already explained how basic patterns represent concepts and best practices inspired by safety standards and guidelines. Each basic pattern addresses a limited number of phenomena. Basic patterns are combined into a composite pattern where the composite addresses all relevant challenges that occur in a specific context. A composite pattern as the one presented in Fig. 1 ease the explanation of how several concepts within conceptual safety design are combined and applied.

Wong et al. [41] reviewed several large development projects and software safety standards from different domains with respect to cost effectiveness and concludes that although standards provide useful and effective guidance, safety and cost effectiveness objectives are successfully met by effective planning and by applying safety engineering best practices evidenced in company best practices throughout the development life cycle. Compared to a standard or a guideline, a composite pattern in the SaCS language may be used to capture such a company specific best practice. In order to accommodate different situations, different compositions of patterns may be defined.

OA.3 The language must be usable without the need of costly tools.

Every pattern used in the cases described in [17][18] was interpreted and applied in its context by a single researcher with background from safety engineering. A conceptual safety design was produced for each case. Every illustration in [5][17][18] and in this paper is created with a standard drawing tool.

V. Related Work

In the literature, pattern approaches supporting development of safety critical systems are poorly represented. In the following we shortly discuss some different pattern approaches and their relevancy to the development of conceptual safety designs.

Jackson [42] presents the problem frames approach for requirements analysis and elicitation. Although the problem frames approach is useful for detailing and analysing a problem and thereby detailing requirements, the problem classes presented in [42] are defined on a very high level of abstraction.

The use of boilerplates [43][44] for requirement specification is a form of requirement templates but nonetheless touches upon the concept of patterns. The boilerplate approach helps the user phrase requirements in a uniform manner and to detail these sufficiently. Although boilerplates may be useful for requirement specification, the focus in SaCS is more towards supporting requirement elicitation and the understanding of the challenges that appear in a specific context.

Withall [45] describes 37 requirements patterns for assisting the specification of different types of requirements. The patterns are defined at a low level; the level of a single requirement. The patterns of Withall may be useful, but as with the boilerplates approach, the patterns support more the specification of requirements rather than requirements elicitation.

Patterns on design and architecture of software-based systems are presented in several pattern collections. One of the well-known pattern collections is the one of Gamma et al. [8] on recurring patterns in design of software based systems. Without doubt, the different pattern collections and languages on system design and architecture represent deep insight into effective solutions. However, design choices should be founded on requirements, and otherwise follow well established principles of good design. The choice of applying one design pattern over another should be based on a systematic process of establishing the need in order to avoid design choices being left unmotivated.

The motivations for a specific design choice are founded on the knowledge gained during the development activities applied prior to system design. Gnatz et al. [46] outline the concept of process patterns as a means to address the recurring problems and known solutions to challenges arising during the development process. The patterns of Gnatz et al. are not tailored for development of safety critical systems and thus do not necessarily reflect relevant safety practices. Fowler presents [7] a catalogue of 63 analysis patterns. The patterns do not follow a strict format but represent a body of knowledge on analysis described textually and by supplementary sketches.

While process patterns and analysis patterns may be relevant for assuring that the development process applied is suitable and leads to well informed design choices, Kelly [10] defines patterns supporting safety demonstration in the form of reusable safety case patterns. The patterns expressed are representative for how we want to address the safety demonstration concern.

A challenge is to effectively combine and apply the knowledge on diverse topics captured in different pattern collections and languages. Henninger and Corrêa [47] survey different software pattern practices and states *"software patterns and collections tend to be written to solve specific problems with little to no regard about how the pattern could or should be used with other patterns"*.

Zimmer [48] identifies the need to define relationships between system design patterns in order to efficiently combine them. Noble [49] builds upon the ideas of Zimmer and defines

 ISBN: 978-1-61208-343-8

a number of relationships such as *uses, refines, used by, combine*, and *sequence of* as a means to define relationships between system design patterns. A challenge with the relations defined by Noble is that they only specify relations on a very high level. The relations do not have the expressiveness for detailing what part of a pattern is *used, refined*, or *combined*. Thus, the approach does not facilitate a precise modelling of relationships.

Bayley and Zhu [50] define a formal language for pattern composition. The authors argue that design patterns are almost always to be found composed with each other and that the correct applications of patterns thus relies on precise definition of the compositions. A set of six operators is defined for the purpose of defining pattern compositions. The language is exemplified on the formalisation of the relationships expressed between software design patterns described by Gamma et al. [8]. As we want the patterns expressed in the SaCS language to be understandable to a large community of potential users, we find this approach a bit too rigid.

Smith [51] presents a catalogue of elementary software design patterns in the tradition of Gamma et al. [8] and proposes the Pattern Instance Notation (PIN) for expressing compositions of patterns graphically. The notation uses simple rounded rectangles for abstractly representing a pattern and its associated roles. Connectors define the relationships between patterns. The connectors operate on the defined roles of patterns. The notation is comparable to the UML collaboration notation [52].

UML collaborations [52] are not directly instantiable. Instances of the roles defined in a collaboration that cooperates as defined creates the collaboration. The main purpose is to express how a system of communicating entities collectively accomplishes a task. The notation is particularly suitable for expressing system design patterns.

Several notations [53][54][55] for expressing patterns graphically use UML [52] as its basis. The notations are simple, but target the specification of software.

VI. Conclusion

We have presented an analytical evaluation of the SaCS pattern language with respect to six different appropriateness factors. We arrived at the following conclusions:

- *Domain*: In the design of the SaCS language we have as much as possible selected keywords and icons in the spirit of leading literature within the area. This indicates that we at least are able to represent a significant part of the concepts of relevance for conceptual safety design.
- *Modeller*: There may be relevant tacit knowledge that is not easily externalised as the SaCS language is today. However, the opportunity of increasing the number of basic patterns makes it possible to at least reduce the gap.
- *Participants*: The terms used for concepts have been carefully selected based on leading terminology within safety engineering. The SaCS language facilitates representing the application of best practices within safety design and mirror leading international standards; in particular IEC 61508.
- *Comprehensibility*: The comprehension of individual patterns and pattern compositions is supported by the use of terms commonly applied within the relevant industrial domains as well as by the application of principles of good design in visualisations, such as the Gestalt principles of perception [35][38].
- *Tool*: Tool support may be provided on the basis of the syntax and semantics of the SaCS language [5].
- *Organisational*: Organisations developing safety critical systems are assumed to follow a development process in accordance to what is required by standards. Wong et al. [41] reviewed several large development projects and software safety standards from different domains with respect to cost effectiveness and concludes that although standards provide useful and effective guidance, safety and cost effectiveness objectives are successfully met by effective planning and by applying safety engineering best practices evidenced in company best practices throughout the development life cycle. SaCS patterns may be defined, applied, and combined in a flexible manner to support company best practices and domain specific best practices.

Acknowledgment

This work has been conducted and funded within the OECD Halden Reactor Project, Institute for energy technology (IFE), Halden, Norway. The second author has partly been funded by the ARTEMIS project CONCERTO.

References

[1] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern-Oriented Software Architecture: On Patterns and Pattern Languages. Wiley, 2007, vol. 5.

[2] C. Alexander, S. Ishikawa, and M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977, vol. 2.

[3] J. C. Knight, "Safety Critical Systems: Challenges and Directions," in Proceedings of the 24th International Conference on Software Engineering (ICSE'02). ACM, 2002, pp. 547–550.

[4] J. Krogstie, Model-based Development and Evolution of Information Systems: A Quality Approach. Springer, 2012.

[5] A. A. Hauge and K. Stølen, "Syntax & Semantics of the SaCS Pattern Language," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1052, 2013.

[6] A. Aguiar and G. David, "Patterns for Effectively Documenting Frameworks," in Transactions on Pattern Languages of Programming II, ser. LNCS, J. Noble, R. Johnson, P. Avgeriou, N. Harrison, and U. Zdun, Eds. Springer, 2011, vol. 6510, pp. 79–124.

[7] M. Fowler, Analysis Patterns: Reusable Object Models. Addison-Wesley, 1996.

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

[9] R. S. Hanmer, Patterns for Fault Tolerant Software. Wiley, 2007.

[10] T. P. Kelly, "Arguing Safety – A Systematic Approach to Managing Safety Cases," Ph.D. dissertation, University of York, United Kingdom, 1998.

[11] B. Rubel, "Patterns for Generating a Layered Architecture," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 119–128.

 ISBN: 978-1-61208-343-8

[12] J. Mendling, G. Neumann, and W. van der Aalst, "On the Correlation between Process Model Metrics and Errors," in Proceedings of 26th International Conference on Conceptual Modeling, vol. 83, 2007, pp. 173–178.

[13] A. G. Nysetvold and J. Krogstie, "Assessing Business Process Modeling Languages Using a Generic Quality Framework," in Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05) Workshops. Idea Group, 2005, pp. 545–556.

[14] J. Krogstie and S. D. F. Arnesen, "Assessing Enterprise Modeling Languages Using a Generic Quality Framework," in Information Modeling Methods and Methodologies. Idea Group, 2005, pp. 63–79.

[15] D. L. Moody, G. Sindre, T. Brasethvik, and A. Sølvberg, "Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework," in Proceedings of the 21st International Conference on Conceptual Modeling, ser. LNCS. Springer, 2013, vol. 2503, pp. 380–396.

[16] J. Becker, M. Rosemann, and C. von Uthmann, "Guidelines of Business Process Modeling," in Business Process Management, ser. LNCS, vol. 1806. Springer, 2000, pp. 30–49.

[17] A. A. Hauge and K. Stølen, "A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Nuclear Power Production," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1029, 2013.

[18] ——, "A Pattern-based Method for Safe Control Conceptualisation – Exemplified Within Railway Signalling," Institute for energy technology, OECD Halden Reactor Project, Halden, Norway, Tech. Rep. HWR-1037, 2013.

[19] I. Hogganvik, "A Graphical Approach to Security Risk Analysis," Ph.D. dissertation, Faculty of Mathematics and Natural Sciences, University of Oslo, 2007.

[20] I. Habli and T. Kelly, "Process and Product Certification Arguments – Getting the Balance Right," SIGBED Review, vol. 3, no. 4, 2006, pp. 1–8.

[21] The members of the Task Force on Safety Critical Software, "Licensing of safety critical software for nuclear reactors: Common position of seven european nuclear regulators and authorised technical support organisations," http://www.belv.be/, 2013, [accessed: 2014-04-10].

[22] C. Haddon-Cave, "The Nimrod Review: An independent review into the broader issues surrounding the loss of the RAF Nimrod MR2 aircraft XV230 in Afghanistan in 2006," The Stationery Office (TSO), Tech. Rep. 1025 2008-09, 2009.

[23] IEC, "IEC 61508 Functional safety of electrical/electronic/programmble electronic safety-related systems, 2nd edition," International Electrotechnical Commission, 2010.

[24] CENELEC, "EN 50129 Railway applications – Communications, signalling and processing systems – Safety related electronic systems for signalling," European Committee for Electrotechnical Standardization, 2003.

[25] European Commission, "Commission Regulation (EC) No 352/2009 on the Adoption of Common Safety Method on Risk Evaluation and Assessment," Official Journal of the European Union, 2009.

[26] ERA, "Guide for the Application of the Commission Regulation on the Adoption of a Common Safety Method on Risk Evaluation and Assessment as Referred to in Article 6(3)(a) of the Railway Safety Directive," European Railway Agency, 2009.

[27] IEC, "IEC 61513 Nuclear power plants – Instrumentation and control systems important to safety – General requirements for systems, 2nd edition," International Electrotechnical Commission, 2001.

[28] A. Ratzka, "User Interface Patterns for Multimodal Interaction," in Transactions on Pattern Languages of Programming III, ser. LNCS, J. Noble, R. Johnson, U. Zdun, and E. Wallingford, Eds. Springer, 2013, vol. 7840, pp. 111–167.

[29] D. Riehle and H. Züllinghoven, "A Pattern Language for Tool Construction and Integration Based on the Tools and Materials Metaphor," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 9–42.

[30] K. Wolf and C. Liu, "New Client with Old Servers: A Pattern Language for Client/Server Frameworks," in Pattern Languages of Program Design, J. Coplien and D. Schmidt, Eds. Addison-Wesley, 1995, pp. 51–64.

[31] IEC, "IEC 61025 Fault Tree Analysis (FTA), 2nd edition," International Electrotechnical Commission, 2006.

[32] ——, "IEC 60812 Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA), 2nd edition," International Electrotechnical Commission, 2006.

[33] N. Storey, Safety-critical Computer Systems. Prentice Hall, 1996.

[34] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," Psychological Review, vol. 63, no. 2, 1956, pp. 81–97.

[35] W. Lidwell, K. Holden, and J. Butler, Universal Principles of Design, 2nd ed. Rockport Publishers, 2010.

[36] J. H. Larkin and H. A. Simon, "Why a Diagram is (Sometimes) Worth Ten Thousand Words," Cognitive Science, vol. 11, no. 1, 1987, pp. 65–100.

[37] W. D. Ellis, A Source Book of Gestalt Psychology. The Gestalt Journal Press, 1997.

[38] M. Wertheimer, "Laws of Organization in Perceptual Forms," in A sourcebook of Gestalt Psychology, W. D. Ellis, Ed. Routledge and Kegan Paul, 1938, pp. 71–88.

[39] J. Wagemans, J. H. Elder, M. Kubovy, S. E. Palmer, M. A. Peterson, M. Singh, and R. von der Heydt, "A Century of Gestalt Psychology in Visual Perception: I. Perceptual Grouping and Figure-Ground Organization," Psychological Bulletin, vol. 138, no. 6, 2012, pp. 1172–1217.

[40] ISO/IEC, "14977:1996(E) Information technology - Syntactic metalanguage - Extended BNF," International Organization for Standardization / International Electrotechnical Commission, 1996.

[41] W. E. Wong, A. Demel, V. Debroy, and M. F. Siok, "Safe Software: Does It Cost More to Develop?" in Fifth International Conference on Secure Software Integration and Reliability Improvement (SSIRI'11), 2011, pp. 198–207.

[42] M. Jackson, Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, 2001.

[43] E. Hull, K. Jackson, and J. Dick, Requirements Engineering, 3rd ed. Springer, 2010.

[44] G. Sindre, "Boilerplates for application interoperability requirements," in Proceedings of 19th Norsk konferanse for organisasjoners bruk av IT (NOKOBIT'12). Tapir, 2012.

[45] S. Withall, Software Requirement Patterns (Best Practices), 1st ed. Microsoft Press, 2007.

[46] M. Gnatz, F. Marschall, G. Popp, A. Rausch, and W. Schwerin, "Towards a Living Software Development Process based on Process Patterns," in Proceedings of the 8th European Workshop on Software Process Technology (EWSPT'01), ser. LNCS, vol. 2077. Springer, 2001, pp. 182–202.

[47] S. Henninger and V. Corrêa, "Software Pattern Communities: Current Practices and Challenges," in Proceedings of the 14th Conference on Pattern Languages of Programs (PLOP'07). ACM, 2007, pp. 14:1–14:19, article No. 14.

[48] W. Zimmer, "Relationships between Design Patterns," in Pattern Languages of Program Design. Addison-Wesley, 1994, pp. 345–364.

[49] J. Noble, "Classifying Relationships between Object-Oriented Design Patterns," in Proceedings of Australian Software Engineering Conference (ASWEC'98), 1998, pp. 98–107.

[50] I. Bayley and H. Zhu, "A Formal Language for the Expression of Pattern Compositions," International Journal on Advances in Software, vol. 4, no. 3, 2012, pp. 354–366.

[51] J. M. Smith, Elemental Design Patterns. Addison-Wesley, 2012.

[52] OMG, "Unified Modeling Language Specification, Version 2.4.1," Object Management Group, 2012.

[53] H. Byelas and A. Telea, "Visualization of Areas of Interest in Software Architecture Diagrams," in Proceedings of the 2006 ACM Symposium on Software Visualization (SoftVis'06), 2006, pp. 105–114.

[54] J. Dong, S. Yang, and K. Zhang, "Visualizing Design Patterns in Their Applications and Compositions," IEEE Transactions on Software Engineering, vol. 33, no. 7, 2007, pp. 433–453.

[55] J. M. Vlissides, "Notation, Notation, Notation," C++ Report, 1998, pp. 48–51.

 ISBN: 978-1-61208-343-8