Visual notations for software pattern languages: a mapping study

Conference Paper · September 2018

DOI: 10.1145/3266237.3266266

3 authors, including:

Monalessa Perini Barcellos
Universidade Federal do Espírito Santo
76 PUBLICATIONS **295** CITATIONS
SEE PROFILE

Ricardo de Almeida Falbo
Universidade Federal do Espírito Santo
179 PUBLICATIONS **1,818** CITATIONS
SEE PROFILE

Visual Notations for Software Pattern Languages: a Mapping Study

Glaice Kelly da Silva Quirino
Mechanical Technician Department
Federal Institute of Espírito Santo
Aracruz, ES, Brazil
glaice.monfardini@ifes.edu.br

Monalessa Perini Barcellos
Ontology and Conceptual Modeling Research Group (NEMO) - Computer Science Department
Federal University of Espírito Santo
Vitória, ES, Brazil
monalessa@inf.ufes.br

Ricardo de Almeida Falbo
Ontology and Conceptual Modeling Research Group (NEMO) - Computer Science Department
Federal University of Espírito Santo
Vitória, ES, Brazil
falbo@inf.ufes.br

ABSTRACT

Reuse has been recognized as an important practice in software engineering. The use of patterns makes it easier to reuse successful solutions, speeds up the development process, and promotes the application of good practices. Related patterns can be organized in a Pattern Language (PL), which represents the patterns and their relations, and provides guidance on how to select, reuse and integrate them. Visual notations are often used to provide a graphical representation to PLs. Aiming to investigate how PLs related to software have been visually represented, we carried out a systematic mapping. We identified and analyzed 64 PLs. As a result, we noticed a lack of consensus on the elements that should be represented in a PL and the symbols used to represent them. Moreover, most PLs have ambiguous or inexpressive visual representations.

CCS CONCEPTS

• **Software and its engineering → Reusability**; Design patterns;

KEYWORDS

Pattern Language, Visual Notation, Mapping Study

ACM Reference Format:
Glaice Kelly da Silva Quirino, Monalessa Perini Barcellos, and Ricardo de Almeida Falbo. 2018. Visual Notations for Software Pattern Languages: a Mapping Study. In *XXXII BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING (SBES 2018), September 17–21, 2018, Sao Carlos, Brazil*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3266237.3266266

1 INTRODUCTION

Patterns are vehicles for encapsulating knowledge. They are considered one of the most effective means for naming, organizing, and reasoning about design knowledge. According to Buschmann et al. [11], a pattern describes a particular recurring design problem that arises in a specific design context and presents a well-proven solution for a problem. The solution is specified by describing the roles of its constituent participants, their responsibilities and relationships, and the ways in which they collaborate.

Software patterns are attempts to describe successful solutions to common software problems [48]. They reflect common structures of these solutions, and can be applied over and over again, when developing software applications in particular contexts. There are several types of software patterns, such as analysis patterns, architectural patterns, design patterns, programming patterns (or idioms) and process patterns [14].

A pattern is a well-succeeded solution for a specific problem. However, in general, a single pattern is not enough to provide the solution to a larger problem. In such case, it is necessary to combine several patterns [19]. In this context, a Pattern Language (PL) is an alternative to provide guidelines that show how patterns can be used together to form solutions to larger problems. PLs contribute to the use of patterns and, consequently, to reuse [19].

Although patterns have been successfully documented for over two decades, building pattern languages has proved to be a more difficult task [24]. Using visual notations, i.e., those using graphical symbols, is helpful, since they allow representing graphically the patterns and their relations, providing a whole view of the PL and contributing to its understanding. Visual notations play an important role in communication because of their effectiveness in transmitting information [40]. According to Ware [52], people acquire more information through vision than through all other senses combined. In addition, diagrams can convey information more concisely and accurately than natural language [39].

Given the benefits of using PLs and the importance of a quality visual notation for representing them, we carried out a mapping study to investigate how software PLs have been visually represented. Before performing our study, we searched for secondary studies addressing visual notations of software PLs. Since we did not find any, we decided carried out the study addressed in this paper. A mapping study is a secondary study designed to give an overview of a research area through classification and counting contributions in relation to the categories of that classification. It makes a broad study in a topic of a specific theme and aims to identify available evidence about that topic [32, 44]. Moreover, the panorama provided by a mapping study allows identifying issues in the researched topic that should be addressed in future research. In our study, we analyzed the visual notation of 64 software PLs. As a result, we noticed a lack of consensus on what should be represented in a PL and how to represent the PL elements. More than

SBES 2018, September 17–21, 2018, Sao Carlos, Brazil

ACM ISBN 978-1-4503-6503-1/18/09…$15.00
https://doi.org/10.1145/3266237.3266266

that, we noticed that most of the visual representations have expressivity problems, such as symbol overload or deficit. The findings point out to the need to improve the quality of the visual notations, in order to make the use of software PLs easier and more efficient, potentializing reuse.

This paper is organized as follows. Section 2 introduces the theoretical foundations on pattern languages and visual notations. Section 3 presents the followed research method and the research protocol used in the study. Results are presented in Section 4. Section 5 discusses the results, their implications, and limitations. Section 6 addresses the study limitations, and, finally, Section 7 concludes the paper and presents directions for future work.

2 PATTERN LANGUAGE AND VISUAL NOTATION

A pattern is a recurring solution to a standard problem [48]. Patterns exist in several phases of software development, and for several purposes. Analysis patterns are reusable models resulting from conceptual modeling activities and apply to common business problems. Architectural patterns are concerned with the overall structure of a software (sub)system, and represent selected types of components and connectors, together with a control structure that governs the overall functioning of the (sub)system. Design patterns abstract and identify the key aspects of common design structures. They are low-level abstractions in the sense that they are concerned with classes, their instances, and relationships. Programming patterns (or idioms) are specific to a programming language, representing commonly accepted programming styles, including how to implement particular behavior or structures in code using the features of the given language. Organizational and process patterns apply to software development processes and organizational pragmatics, i.e., to software developers and users, relationships between them, and relationships between people and software. These are the types of software patterns most commonly found in the literature. However, there are also other types of software patterns [14].

Patterns are often defined and applied separately. However, no pattern is an island. Contrariwise, patterns are fond of company: sometimes with one pattern as an alternative to another, sometimes with one pattern as an adjunct to another, sometimes with a number of patterns bound together as a tightly-knit group. The manifold relationships that can exist among patterns help to strengthen and extend the power of an individual pattern beyond its specific focus. Therefore, a broader context is needed to describe how patterns can be used together to solve larger problems and how to solve new problems that can arise when patterns are used in combination. This context is provided by a Pattern Language (PL) [11].

A PL is a set of patterns and relationships among them that can be used to systematically solve coarse-grained problems [13]. A PL defines a process that aims to provide holistic support for using the patterns to address problems related to a technical domain or application. This holistic view should provide explicit guidance on problems that may arise, inform the possible means of solving them and suggest one or more patterns to solve each specific problem [11].

PLs reflect the fact that patterns tend to be strongly coupled and it is difficult or even impossible to use them in isolation [17]. While

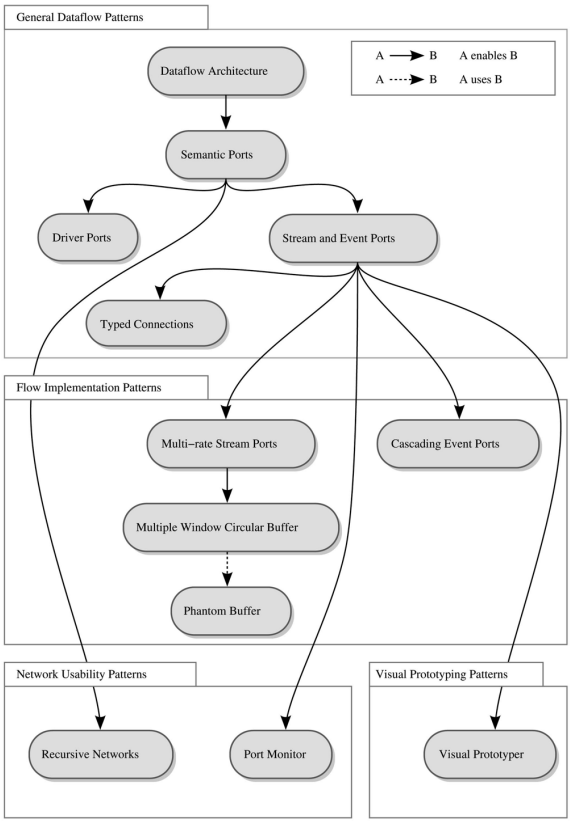

Figure 1: The multimedia dataflow pattern language.

single patterns aim at solving separate problems, a PL is considered as a constructive guide to combine patterns together to resolve more complex problems, enabling developers to use patterns together to meet their needs [38].

For a software PL to be effective in its goal of guiding developers in the application of the patterns and adequately handling the combined application of several patterns, it is necessary that the relationships among the patterns are clearly defined and explained in the PL.

Alexander et al. [2] propose the use of diagrams to represent the relationship among patterns. Therefore, visual notations can be used to graphically represent PLs. The purpose of adopting visual notations is to give an overview of patterns and their relationships to provide holistic understanding of the PL and assist in patterns selection. Figure 1 presents an example of graphical representation of PL. The PL shown in the figure contains patterns to aid in the development of multimedia processing systems [3].

Visual notations encode information using spatial arrangements of graphical elements. They are used in several areas for creating graphical representations that serve as abstractions capable of helping understand the information that one wishes to communicate. According to Moody [39], visual notations are effective because they are processed in parallel by the human visual system. About a quarter of the human brain is dedicated to vision, which is more than all other senses combined.

In his work entitled The Physics of Notation (PoN) [39], Moody defines a set of principles for designing cognitively effective visual notations. PoN considers information visualization and pragmatic theories to improve the cognitive effectiveness of visual notations. PoN identifies nine principles for designing cognitively effective visual notations, namely: (i) *Semiotic Clarity:* "There should be a 1:1 correspondence between a meta-model construct and a graphical symbol"; (ii) *Semantic Transparency:* "Use symbols whose appearance suggests their meaning"; (iii) *Perceptual Discriminability:* "Symbols should be clearly distinguishable from one another"; (iv) *Complexity Management:* "Include explicit mechanisms for dealing with complexity"; (v) *Cognitive Integration:* "Include explicit mechanisms to support integration of information from different diagrams"; (vi) *Visual Expressiveness:* "Use the full range and capacities of visual variables"; (vii) *Graphic Economy:* "The number of different graphical symbols should be cognitively manageable"; (viii) Dual Coding: "Use text to complement graphics"; *(ix) Cognitive Fit:* "Use different visual dialects for different tasks and audiences" [39].

Over the years, several PLs have been created and visually represented. As a consequence, a number of different visual notations have emerged. Aiming to investigate the visual notations that have been used to represent software PLs, we performed the mapping study presented in this paper.

3 THE RESEARCH PROTOCOL

The study was performed following the approach defined by Kitchenham and Charters [32]. According to this approach, a systematic mapping involves: planning, when the research protocol is defined; conducting, when the protocol is executed and data are extracted, analyzed, and recorded; and reporting, when the results are recorded and made available to potential interested parties. In this section we present the main parts of the research protocol used in the study.

The study **goal** is to investigate visual notations adopted to represent software PLs. For achieving this goal, we defined nine research questions that are shown in Table 1.

The **search string** adopted in the study is composed of terms related to PL and software. The following search string was used: *("pattern language" OR "patterns language") AND ("software").*

For establishing this search string, we performed some tests using different terms, logical connectors, and combinations among them. More restrictive strings excluded some important publications identified during the informal literature review that preceded the mapping study. These publications were used as control publications, meaning that the search string should be able to retrieve them. We decided to use a comprehensive string that provided better results in terms of number and relevance of the selected publications, even thought it had returned many publications that had to be eliminated in subsequent steps.

The search was performed in the following six **sources**, selected based on other systematic reviews in the Software Engineering area: IEEE Xplore, ACM Digital Library, Scopus, Science Direct, Engineering Village and Web of Science. For searching these sources, the search string went through syntactic adaptations according to particularities of each digital library engine.

Publications selection was performed in four steps. In *Preliminary Selection and Cataloging (S1)*, the search string was applied in the search mechanism of each digital library (we limited the search scope to title, abstract and keywords metadata fields). After that, in *Duplications Removal (S2)*, publications indexed in more than one digital library were identified and duplications were removed. In *Selection of Relevant Publications - 1st filter (S3)*, the abstracts of the selected publications were analyzed considering the following inclusion (IC) and exclusion (EC) criteria: (IC1) The study proposes or apply a software PL; (EC1) the publication does not have an abstract; (EC2) the study was published only as an abstract; (EC3) the publication is not written in English; (EC4) the publication is a secondary study, a tertiary study, a summary, an editorial or a tutorial. In *Selection of Relevant Publications - 2nd filter (S4)*, the full texts of the publications selected in S2 were read and analyzed considering the cited inclusion and exclusion criteria. In this step, aiming to focus on publications addressing software PLs with a visual representation, we applied the following additional inclusion criterion: (IC2) the PL described in the publication has a visual (i.e., a graphical) representation. Moreover, to avoid study repetition, we considered another exclusion criterion: (EC5) the publication is a copy or an older version of an already selected publication. Publications whose full text was not available were also excluded (EC6).

The publications returned in the publication selection steps were cataloged and stored in spreadsheets. We defined an id for each publication and recorded the publication title, authors, year, and vehicle of publication. Data from the publications returned in S4 were extracted and organized into a data extraction form oriented to the research questions. The spreadsheets and form produced during the study can be found in [4].

The first author performed publication selection and data extraction. The second and third authors reviewed both. Once data has been validated, the first author carried out data interpretation and analysis, and again the second and third authors reviewed the results. Discordances were discussed and resolved in meetings. Quantitative data were tabulated and used in graphs and statistical analysis. Finally, the three authors performed qualitative analysis considering the findings, their relation to the research questions and the systematic mapping purpose.

4 DATA EXTRACTION AND SYNTHESIS

The mapping study considered papers published until December 2017. Searches were conducted for the last time in March 2018. Figure 2 illustrates the followed process and the number of publications selected in each step.

In the 1st step, as a result of searching the selected sources, a total of 1,194 publications was returned. In the 2nd step, we eliminated duplications, achieving 546 publications (reduction of approximately 47%). In the 3rd step, we applied the selection criteria (inclusion and exclusion criteria) over title, abstract and keywords, resulting in 258 papers (reduction of approximately 47%). It is important to emphasize that, at this step, we only excluded publications that were clearly unrelated to the subject of interest, as suggested by Kitchenham and Charters [32]. In case of doubt, the paper was taken to the next step. In the 4rd step, the selection criteria were applied

Table 1: Research questions and their rationales.

ID	Research Question	Rationale
RQ1	When and in which type of vehicle have the studies been published?	Aims at giving an understanding on when and where (journal/conference/symposium /workshop) publications about software PLs with visual representation have been published.
RQ2	Which types of research have been done?	Investigates which type of research is reported in each selected publication. We consider the classification defined by Wieringa et al. [54], which includes the following categories: Evaluation Research; Proposal of Solution; Validation Research; Philosophical Paper; Opinion Paper; and Personal Experience Paper. This question is useful to evaluate the maturity stage of the research topic.
RQ3	Are the software PLs of general or specific purpose?	Investigates whether PLs have been defined for use in specific application domains (e.g., financial, health, etc.), classes of systems (e.g., web system, agent-based system, etc.) or if they can be used for any domain or class of system.
RQ4	What types of patterns are found in the software PLs?	Identifies the types of patterns (e.g., analysis patterns, design patterns, process patterns, etc.) represented in the PLs and verifies if there has been predominance of certain types.
RQ5	What elements are addressed in the software PLs and what symbols have been used to represent them?	Identifies the elements represented in the PLs (e.g., patterns, relations, patterns groups) and the adopted symbols. This question allows analyzing if there is consensus on the elements being represented and the symbols used to represent them.
RQ6	Do the software PLs define the types of relationship? If so, what types of relationship are defined?	Identifies whether PLs explicitly define the types of relationships between patterns and which types are defined (e.g., dependency, variant of, etc.). This question allows analyzing if PLs have been clear about the relationships between patterns and if there has been predominance of certain types of relationship.
RQ7	How are patterns grouped in software PLs and how are groupings represented?	Investigates whether there is some sort of grouping of patterns in the PLs, which criteria for grouping are used, and how the groups are represented. This question is useful to evaluate if visual notations have been concerned with managing complexity.
RQ8	In addition to the visual representation, what mechanism is provided to guide patterns selection?	Investigates whether software PLs provide some additional mechanism to guide patterns selection (e.g., some sort of description). This question helps verify if there is a concern with providing clear guidance for users to select patterns.
RQ9	Which are the types of models provided by software PLs?	Identifies whether PLs provide structural models (which represent the elements of the PLs and the relationships between them), process models (which represent flows that indicate the possible paths for selecting patterns) or both. This question allows verifying whether PLs have been visually represented considering different and complementary perspectives.

considering the full text, resulting in 64 publications (reduction of approximately 75%). Due to space limitation, these publications are not listed in this paper. Along this and the next section we refer to some of the selected publications. The complete list, including a summary about each publication, can be found in [4].

Publication year and vehicle (RQ1): Figure 3 shows the distribution of the 64 selected publications over the year, providing a general view of the efforts in the area of software PLs with visual representation. As shown in this figure, the first publication of a software PL with visual representation occurred in 1998. Figure 3 also shows the distribution of publications considering the vehicle of publication. Papers have been published in three types of vehicles: Journals, Conferences and Workshops. Conferences have been the main forum, encompassing 62.5% of the publications (40 out of 64), followed by journals, publishing 32.8% of the papers (21 out of 64), and workshops, with only three publications (4.7%). Taking the period of publications into account, we can notice a long-term effort regarding the development of software PLs, since this topic has been the target of researchers for almost 20 years. However, the

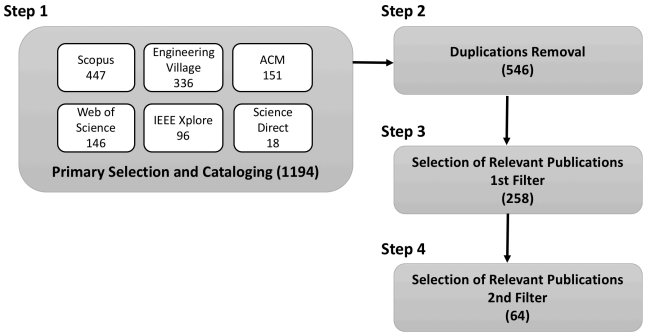

Figure 2: Publication Selection Process

results have been mostly published in conferences (62.5%). The low percentage of journal publications, which generally require more mature works, can be seen as a sign that the research on this topic is not mature enough yet.

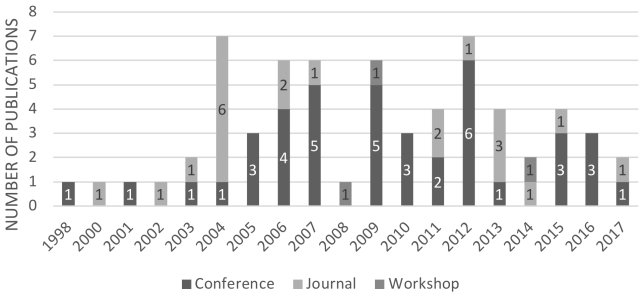

Figure 3: Publications over the years

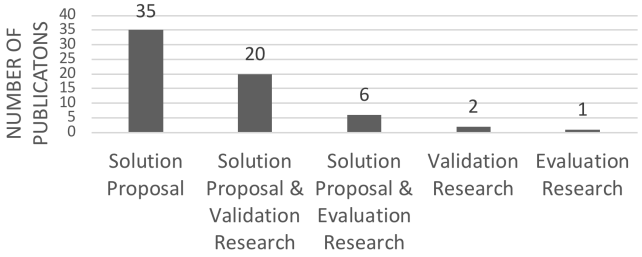

Figure 4: Types of Research

Research Type (RQ2): 61 out 64 publications propose a solution for a problem and argue for its relevance. Thus, they were classified as *Proposal of Solution*. 26 of these publications also report some kind of evaluation, being 20 (31.3%) *Validation Research* (investigates the characteristics of a proposed solution not yet implemented in practice) and six (9.4%) *Evaluation Research* (discusses the implementation and evaluation of a proposed solution in an industrial setting). The [21, 34, 45] are some of the works that validated the proposed PL. [18, 20, 51], in turn, are examples of works that propose a software PL and evaluate its application in a case study in the industry. Only one publication, [49], (1.5%) refers exclusively to Evaluation Research, while two, [41, 56], (3.1%) refer exclusively to Validation Research. These results show that only around 46% of the efforts presented in the publications were concerned with some sort of evaluation, and just in few cases the proposal reached the industry. This reinforces the results from RQ1 that research on this topic is not mature enough yet. Figure 4 shows the types of research identified in the study.

PLs Purpose (RQ3): Among the PLs selected in the study, 26 (41%) have *general purpose*, and can be used independently of the application domain or class of system. However, most PLs, 38 (59%), were conceived with *specific purpose*, i.e., to a specific application domain or class of system. The PL presented in [12] is an example of a general purpose PL. Its patterns can be applied to produce secure software regardless the application domain or class of system. On the other hand, the PL presented in [18], which deals with aspects related to accessibility in web systems, has a specific purpose, since although it is domain-independent, it is restrict to a specific class of system (web systems). The PL presented in [49] to assist in the development of banking systems is also an example of PL with

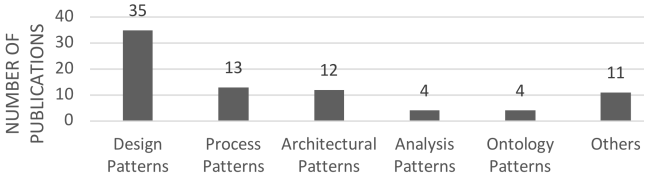

Figure 5: Types of patterns contained in PLs

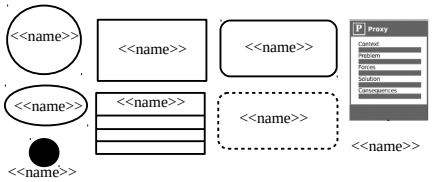

Figure 6: Different symbols for Pattern

specific purpose. It is independent of class of system, but it is for a specific domain (banking).

Type of Patterns (RQ4): Several types of patterns were addressed by the analyzed PLs, including: *Design Patterns* (addressed in 35 of the 64 publications - 54.7%), *Process Patterns* (13 publications - 20.3%), *Architectural Patterns* (12 publications - 18.8%), *Analysis Patterns* (4 publications - 6.2%) and *Ontology Patterns* (4 publications - 6.2%). Other types of patterns, such as *Access Patterns, Instructional Patterns, Interaction Patterns,* among others, were addressed by only one PL. These patterns were grouped under the category *Other* (11 publications - 17.2%). Some PLs addressed more than one type of pattern, and therefore the sum of the percentages exceeds 100%. Figure 5 shows the types of patterns found in the study.

Types of Element and their Symbols (RQ5): All the selected publications contain some visual representation for the PL. In some cases, the elements of the visual notation are explicitly defined. For example, in [7], a legend presents the symbols of the elements used in the visual representation and their meaning. In other cases, such as in [22], only the visual representation is shown, without explicit information about the respective elements. In such cases, we analyzed the publication text to obtain information about the elements of the visual representation.

The visual representation of some PLs contains several elements (e.g., [16]), while others (e.g., [9, 28, 31, 53]) contain only *Pattern* and *Flow*, which are the most basic elements for representing a PL. Moreover, although several visual representations have the same element (e.g., all visual representations analyzed have the Pattern element), different symbols are used to represent it. For example, we identified 9 different symbols used to represent Pattern (see Figure 6). Despite the variety of symbols, in the visual representations, there is a predominance of geometric symbols. Figure 7 shows the elements identified in the visual representations analyzed in the study and the number of different symbols we found representing each element.

Types of relationship (RQ6): The elements of a PL can relate to each other in different manners and it is important to understand how they relate. Concerning how the types of relationship are defined, in 40 (62.5%) of the analyzed publications, the authors

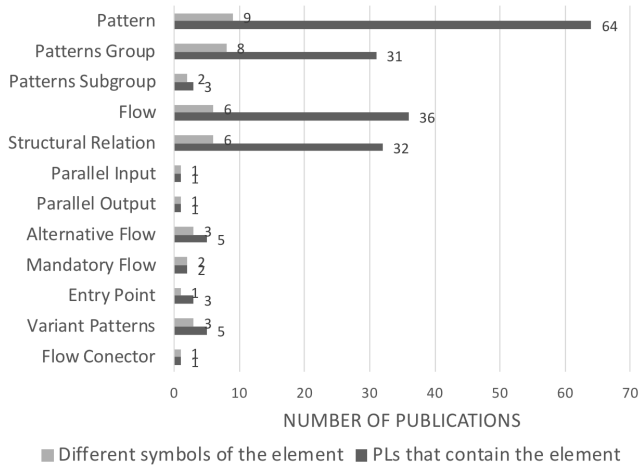

Figure 7: Elements and symbols of the PLs

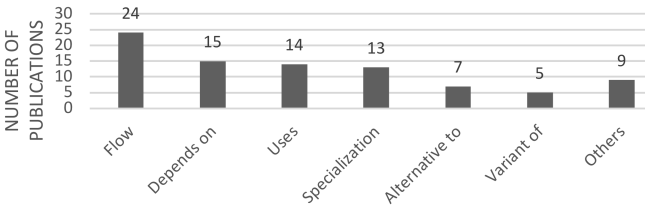

Figure 8: Types of relationship

explicitly define the types of relationship (e.g., dependency, use, variance), such as in [12, 15]. Nineteen (29.7%) of the publications (e.g., [31, 42, 49]) *do not define* the types of relationship and only connect elements (for example, using an arrow that connects one pattern to another) to indicate that they are related in some way . Five (7.8%) of the publications *partially define* the types of relationship, since only some of the types of relationship used in the visual representation are defined (e.g., [27, 59]).

Regarding the types of relationship, considering those explicitly defined in the publications, the main types identified are: *Flow* (indicates the admissible sequences in which patterns can be applied); *Depends on* (indicates a structural dependency between the patterns); *Uses* (indicates that one pattern uses another pattern when the second pattern solves a problem probably raised by the application of the first); *Specialization* (indicates that one pattern can be a specialization of the solution provided by another pattern); *Alternative to* (indicates patterns that can be used alternatively); and *Variant of* (indicates mutually exclusive patterns that solve the same problem but in different ways). Other types appear less frequently, such as *Implements* and *Conflicts*, and were categorized as *Others*. Flow is the predominant type of relationship in the analyzed PLs. Some PLs (e.g., [8, 35, 51]) have only this type of relationship, while others also contain other types (e.g., [24, 35]). Figure 8 shows the types of relationship identified in the analyzed PLs.

It is possible to notice that some elements identified in RQ5 are related to types of relationship. For example, the Structural Relations element identified in RQ5 (see Figure 7) encompasses several types of relationship identified in RQ6, such as Depends on, Uses and Specialization. However, in RQ5, we considered all the elements included in the visual representations of the PLs, while in RQ6 we considered only the types of relationship explicitly defined in the PLs. Thus, there is difference between the number of publications related to the same element in RQ5 and RQ6. Flow, for example, was found in 36 publications (see Figure 7), but only 24 of them (see Figure 8) explicitly defined this type of relation.

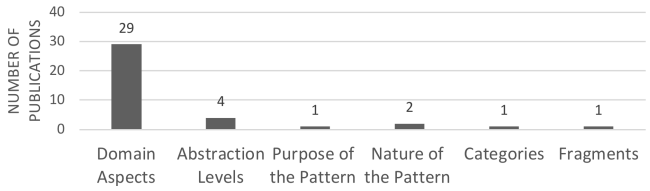

Figure 9: Criteria for grouping patterns

Pattern Grouping (RQ7): Groupings were found in 35 of the analyzed PLs. In most of them, 29, patterns are grouped according to *aspects of the domain*. This can be observed, for example, in [7], which presents an ontology pattern language for software measurement. In this work, patterns are grouped into six groups: Measurable Entities, Measures, Measurement Units & Scales, Measurement Procedures, Measurement Planning, and Measurement & Analysis. In four PLs ([5, 18, 29, 50]), the patterns are grouped considering *levels of abstraction*. Another aspect considered for grouping patterns is the *nature of the problem*. This can be observed in two PLs, presented in [8] and [34]. The last groups patterns into: Architecture, Configuration, Reliability, Memory, Notification, Topology, and Networks. There are also PLs containing groups according to *categories* ([23]), *purpose* ([10]) and *fragments* [43]. These types of groupings were found in only one PL. Figure 9 shows the criteria for grouping patterns adopted in the analyzed PLs. A same PL can adopt more than one criterion for grouping patterns, as occurs in [18, 29, 50], which group patterns according to *domain aspects* and *abstraction levels*. Thus, the sum of the occurrences in Figure 9 is greater than 35 which is the number of PLs with pattern groups.

Groups are represented in different ways in the analyzed PLs. *Rectangle* is the most common symbol used to represent groups. PLs presented in [22, 37, 47] are examples of PLs that use this representation. Other PLs (e.g., [26, 27, 42, 49]) use a *rectangle with rounded corners*. Some PLs have represented groupings with other symbols: *colors/textures* ([1, 5, 29, 50]), *UML package symbol* ([6, 42]), *irregular region* ([34, 36]), *partition wrenches* ([10, 18]), *division lines* ([18, 56]), and *ellipse* ([8]). Figure 10 shows the different symbols used to represent patterns groups in the analyzed PLs. Six publications ([23, 28, 35, 43, 51, 59]) report they group patterns, but groups were not presented in the visual notation provided in the publication. For this reason, these studies were disregarded in Figure 10.

Mechanism for Pattern Selection (RQ8): The main purpose of a PL is to provide guidance for patterns selection. Among the PLs analyzed, 50 publications (78%) do not provide any additional mechanism to support patterns selection beyond the visual representation. Fifteen publications (22%) provide a *process description* to

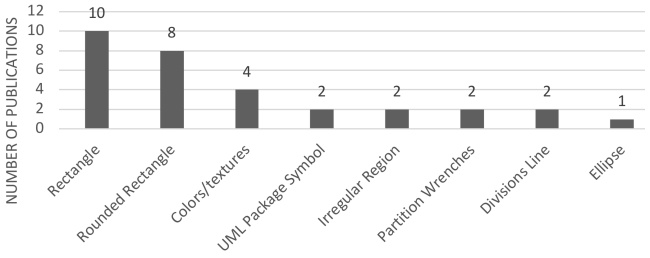

Figure 10: Symbols used to represent groups in PLs

guide patterns selection. For example, the PL presented in [46] has a description of the visual representation that guides pattern selection according to the problems to be addressed. This description enhances the understanding on how patterns should be selected and applied, increasing the effectiveness of PL. One of the 15 publications ([57]) provides, in addition to a process description, a *Feature Model* to assist in choosing the sequence of patterns to be applied. From the 15 publications that provide additional mechanism for pattern selection, only a few offer descriptions clear and detailed enough to guide pattern selection. [7, 16, 46, 56, 58] are examples of works that provide detailed descriptions containing guidelines that consider the possible situations to apply the patterns and indicates the pattern(s) to be selected.

Type of Models (RQ9): Fifty per cent (32 out 64) of the analyzed PLs provide *process models* that indicate the possible flows to be followed for pattern selection (e.g., [9, 24, 51]). Forty-four (28 out 64) present *structural models*, which shows structural relationships (e.g., dependency) between the PL elements (e.g., [25, 47, 53]). The remaining 6% of PLs (four publications) present *both* models. In two of these publications, [41, 43], the visual representation of the PLs is a hybrid model, that is, a single diagram is used to represent both process and structural models. In the other two, [22, 57], two models are used, one to guide the process for selecting and applying patterns, and other to show the structural relationships between the patterns.

5 DISCUSSION

In this section, we provide additional information about the analyzed PLs and make a discussion about the obtained results.

Considering *types of patterns* contained in the analyzed PLs, design patterns are predominant. This means that this type of pattern has been more explored in PLs, which can be understood as a sign that research involving design patterns in software PLs with visual representation are more mature when compared to research involving other types of patterns. For example, we identified only three works addressing ontology patterns [7, 16, 46]. All these works are recent and developed by the same research group.

With respect to the *elements* addressed in the PLs, the results show that they are diverse. Pattern and Flow are the elements addressed in most of the analyzed PLs. Structural relations are also frequently represented. In fact, this was expected, because those are the basic elements to visually represent a PL. Patterns and flows are used to indicate paths to pattern selection, providing a behavioral representation of the PL, while patterns and structural relations are used to provide a structural representation of the PL, showing the structural relations (e.g., dependency) that must be considered to apply the patterns. Although these elements are addressed in several PLs, there are others that appear in only a few of them. This indicates that although there is some agreement on the main elements to be addressed in a PL, there is not consensus on other elements that should be addressed in a PL.

Concerning the *symbols* used to represent the elements, there is a predominance of geometric forms. Although Moody [39] advocates that geometric symbols do not directly demonstrate the element meaning (i.e., they are semantically opaque), they are traditional symbols used in software engineering models and abstractions. Thus, they are familiar to software PLs users and can be representative enough in this context. However, there is no consensus on the symbol to be used to represent each element.

There is also variety in the *number of elements* addressed in the PLs. In [7], for example, seven elements are considered: entry point, pattern, flow, alternative flow, parallel output, parallel input and patterns group. On the other hand, in [58], the PL contains only two elements: pattern and dependency (structural relation). The variation in the number of elements addressed in the PLs may indicate that: (a) PLs treat different problems and some cases need more elements than others; or (b) some PLs use less elements than necessary to provide a clear understanding of the PL. In this case, there is overload in the meaning of the used symbols that represent the elements, which leads to unclear and cognitively poor PLs. During the study, due to the lack of clarity of the PLs visual representation, we faced many difficulties to understand some PLs. This experience leads us to believe that LPs visual representations have been not expressive enough to propitiate proper understanding.

Regarding the *types of relationship*, we notice that most PLs do not always explicitly define the different relationships between the elements. This hinders understanding the relationships between patterns and, consequently, selecting and applying the patterns. In the publications that define types of relationships, there is a predominance of dependency, followed by use and specialization. In fact, a predominance of the dependency relationship in PLs is not surprising, since it can encompass other types of relationship and eliminate (or mask) the need to define them. For example, if a pattern A uses a pattern B, there is a relation of use from A to B (A *uses* B) or, more generally, it can be understood that there is a dependency relation between A and B (A *depends on* B). However, if a PL does not specify that the relationship between A and B is *uses*, it is not possible for users to identify precisely the relation between the patterns. For example, A *depends on* B because A can only be applied after B, since A requires the solution given by B, or A *depends on* B because B is part of the solution proposed in A? In both cases, we can say that A depends on B, but in the second case, the dependency is specifically a relation of use. Some types of relationship, such as complement and implementation, appear only in a few PLs. Some of these types can be defined from others. For example, we can consider that the complement relation could be defined from the use relation. If a pattern A complements a pattern B, we can conclude that B uses A.

As for *pattern grouping*, the results show that this practice has been used in more than 50% of the analyzed PLs. On the one hand, there is a predominance in the criteria used to group patterns (83%

use domain aspects). On the other hand, there is a variety of symbols used to represent groups. A common characteristic to the representations is the use of symbols that delimit a region in which patterns are grouped. Grouping is particularly important in large and complex PLs. Grouping patterns can be a strategy to manage PL complexity. Groups may be represented in a transparent manner (i.e., patterns of the group are visible in the PL) or as black boxes (i.e., the PL only shows the symbol representing the group, without presenting the patterns inside it). In this way, different visions of the PL are provided with different levels of details, contributing to understand the PL. Among the analyzed PLs, only the one presented in [24] represents pattern groups both as black boxes and in the transparent way. Thus, the PL has two possible views: a detailed view presenting patterns and relations between them, and a more general view, showing groups as black boxes.

Concerning the *mechanisms to support pattern selection*, most of the analyzed PLs do not provide any additional mechanism apart from the PL visual representation. The lack of clear guidance on pattern selection compromises PL use. Although visual notations are effective in transmitting information [40], using text as a complementary communication strategy helps better understanding the visual representation [39].

Finally, with respect to *types of model* used to represent the PLs, we observed that in 94% of the publications only one type of model is used, presenting only the behavioral (process) or the structural perspective of the LP. Only four publications represent the LP from different perspectives. Two of them use the same diagram to show both perspectives. This approach tends to hinder the understanding about the model and the perception of the different visions. Using a model to represent the possible paths for pattern selection and application and another to show the structural relationships between the PLs elements can contribute to better understand and use the PL.

Making a general analysis of the visual notations investigated in the study, we observed that they do not satisfy some principles suggested by Moody [39]. Some PLs have overloaded symbols, while others present deficit of symbols. In both cases, the *semiotic clarity* principle is not satisfied, because there is not a 1:1 correspondence between the PL elements and the adopted symbols. In addition, the small visual distance between the symbols in most PLs, and the small number of visual variables explored in the notations compromise *perceptual discriminability* (sometimes it is difficult to distinguish a symbol from another) and *visual expressiveness* (some symbols should use additional variables, such as color or texture, to improve expressiveness).

Despite the lack of compliance with some PoN principles, others are satisfied. In general, the PLs present a cognitively manageable number of different symbols (*graphical economy*) and some of the PLs use text to complement graphics (*double coding*). Besides, although the visual distance between the symbols is small, it is predominant the use of geometric forms, symbolism very common in Software Engineering, which enhances PLs' learning and contributes to meet the *semantic transparency* principle.

Some PLs adopt grouping as a mechanism for dealing with complexity (*complexity management*). However, none of them includes explicit mechanisms to support integration of information from different models (*cognitive integration*). We did not evaluate the *cognitive fit* principle because each PL was developed to only one audience, being unnecessary the use of different visual dialects.

Based on the panorama provided by the study results, we can highlight the following issues: (i) lack of a standard visual notation to represent software PLs (we notice inconsistencies even between visual notations used in works developed by the same research group (e.g., [7, 16, 46])); (ii) visual notations used to represent software PLs are cognitively poor; (iii) lack of concern with additional mechanisms to guide pattern selection; and (iv) lack of a complete view of the PL, showing aspects related to the PL process and structure.

6 LIMITATIONS OF THE STUDY

Although the ambition of a mapping study is to summarize all relevant research in an area, different sets of publications can be obtained given a number of decisions taken. Researchers conducting secondary studies in general have to make a lot of decisions and exercise a lot of judgment. The decisions taken by researchers and the judgments influence the outcome both in terms of which papers are selected and what the researchers conclude from their secondary studies [55]. Thus, as any study, the study presented in this paper has some limitations.

Some challenges can reach the researchers during a mapping study, such as how to select a comprehensive and relevant source of publications, how to consistently apply the inclusion/exclusion criteria, how to classify data and how to interpret them. In this study we experienced these challenges and carried out some actions aiming at minimalizing their influence on the results.

Publication selection and data extraction were initially performed by only one of the authors, and thus some subjectivity could be embedded. To reduce this subjectivity, the other two authors performed these same steps over 47% of the sample containing non-duplicated publications, so that 257 out 546 publications were analyzed by at least two researchers. Discordances and possible biases were discussed in meetings.

An analysis of degree of concordance was performed to measure the level of agreement between the results obtained from the researchers in the selection process. For this, we calculated the kappa coefficient [33]. Kappa is a statistical measure of inter-rater agreement for qualitative (categorical) items. It is considered to be a more robust measure than simple percent agreement calculation, since it takes into account the agreement occurring by chance. Kappa maximum value is 1, representing total agreement. Values close to and below 0 indicate no agreement. Considering the set of publications analyzed by all the reviewers (i.e., 257 publications), the obtained kappa coefficient is 0.77, which, according to Landis and Koch [33], means substantial agreement.

Terminological problems in the search strings may have led to miss some publications. To minimize this possibility, we performed previous simulations in the selected digital libraries and used control publications to establish the string.

Applying the search string to digital libraries engine can also interfere in the returned publications. First, because it is necessary to adapt the string syntax. The adaptation may result in a different search string. The string used in the study is very simple, so we eliminated the risk of making mistakes when adapting it to each

engine. Second, because of problems or changes in the engines. During the study, occurred changes in the engine of one of the used digital libraries. We ran some tests that showed us that the engine interface had changed, but the engine seemed to work as before.

The study considered six digital libraries as sources. They were selected based on other secondary studies in the Software Engineering area. Although this set of digital libraries represents a comprehensive source of publications, the exclusion of other sources and the fact that we did not performe snowballing may have left some valuable publications out of our analysis.

During publication selection, we had to decide if the publications meet our selection criteria. We objectively defined the criteria, however there was still subjectivity when applying them to publications. Aiming to keep consistency in publications selection and avoid misinterpretations about what we should consider as PL, we adopted the argument of Buschmann et al. [11], which says that in pattern catalogs there is a lack of a strong sense of connection between patterns, while in PLs this connection is strongly perceived, and it should be followed when applying the patterns. Based on that, publications presenting pattern catalogs (even when they were incorrectly called PLs) were eliminated.

Another limitation is related to the classifications we made for categorizing data. For each research question we defined a classification schema. Some categories were based on classifications previously proposed in the literature (e.g., type of research [54]). Others were established during data extraction, based on data provided by the analyzed publications (e.g., types of relationship and types of pattern). Determining the categories and how publications fit them involves a lot of judgment. Thus, different results could be obtained by other researchers.

7 CONCLUSIONS

In the 70s, Christopher Alexander [2] presented a pattern language (PL) for towns, buildings and construction. Inspired by Alexander's work, software practitioners and researchers have been organized well-known solutions related to software as software patterns since 1990s. The PLs structure primarily aims at making knowledge easily applicable [30].

Considering the effectiveness of visual notations to transmit information and the consolidated use of models as abstractions in Software Engineering, PLs have been visually represented. In this paper, we presented the main results of a mapping study that investigated software PLs. A total of 1194 publications were considered and the visual representation of 64 software PLs was analyzed.

According to Kitchenham et al. [32] and Wieringa et al. [54], the results of a mapping study help identify gaps that suggest future research. The study results reveal a lack of consensus on what should be represented in a software PL and how to represent the PLs elements. Moreover, most of visual notations used to represent software PLs are cognitively poor and have expressivity problems, such as symbol overload or deficit. There is also lack of concern on providing clear mechanisms to assist in patterns selection, and lack of a complete representation of the PLs, considering behavioral (process) and structural perspectives.

A PL is not merely a set of patterns that solve problems. More than that, a PL should establish a dialog with users guiding them on addressing problems through the reuse of well-proven solutions. The dialog established by PLs is deeper and more comprehensive than the dialog with individual patterns or pattern catalogs[13].

Visual notations are very important tools to establish the dialog between a PL and its users. However, the study results indicate that the visual representations of PLs have not been expressive enough. The findings point out to the need to improve the quality of the visual notations to make the use of PLs easier and more efficient, potentializing reuse.

The lack of a standard to visually represent PLs suggests the need to a visual modeling language to represent software PLs. This modeling language should take aspects of visual communication (e.g., the PoN principles) into account and provide support to represent PL elements in models considering the behavioral and structural perspectives. We did not find any work in this direction in Software Engineering.

As future work, we plan to use the knowledge obtained from the mapping study in the development of a visual modeling language to represent software PLs. We intend to use this language to represent some of the PLs identified in the study and evaluate the new representation against the previous one. We also intend to use the proposed language to represent new software PLs and evaluate the quality of the visual notations produced. We believe that a visual modeling language to represent software PLs is a relevant contribution to the area.

We did not find any work similar to ours. Before performing our study, we searched for secondary studies addressing visual notations of software PLs. We used the following search string: *("pattern language") AND ("systematic literature review" OR "systematic review" OR "systematic mapping" OR "mapping study" OR "systematic literature mapping")*. The search string was applied to three metadata fields (title, abstract and keywords) and in the same digital libraries used in our study. The search returned 7 publications. However, none of them referred to a secondary study addressing our topic of interest.

ACKNOWLEDGMENTS

This research is funded by the Brazilian Research Funding Agency CNPq (407235/2017-5 and 461777/2014-2), CAPES (23038.028816/2016-41), and FAPES (69382549/2014).

REFERENCES

[1] Ademar Aguiar and Gabriel David. 2006. Patterns for Documenting Frameworks-Part II.. In *EuroPLoP*. Citeseer, 393–406.
[2] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. 1977. A pattern language. (1977).
[3] Xavier Amatriain and Pau Arumi. 2011. Frameworks generate domain-specific languages: A case study in the multimedia domain. *Software Engineering, IEEE Transactions* 37, 4 (2011), 544–558.
[4] Authors. 2018. Mapping Study about Visual Notations for Software Pattern Languages. Retrieved April 26, 2018 from https://www.dropbox.com/sh/cfz58ln02ms0fwv/AACWfuhjNiKuqziOQmrDpubka?dl=0
[5] Paris Avgeriou and Peter Tandler. 2006. Architectural patterns for collaborative applications. *International journal of computer applications in technology* 25, 2-3 (2006), 86–101.
[6] Paris Avgeriou and Uwe Zdun. 2005. Architectural patterns revisited-a pattern language. (2005).
[7] Monalessa Perini Barcellos, Ricardo de Almeida Falbo, and Vinícius Frauches. 2014. Towards a Measurement Ontology Pattern Language.. In *ONTO. COM/ODISE@ FOIS*.
[8] D Bellebia and Jean-Michel Douin. 2006. Applying patterns to build a lightweight middleware for embedded systems. In *Proceedings of the 2006 conference on Pattern*

languages of programs. ACM, 29.
[9] Mirla RR Braga, Carla IM Bezerra, José Maria S Monteiro, and Rossana Andrade. 2012. A pattern language for agile software estimation. In *Proceedings of the 9th Latin-American Conference on Pattern Languages of Programming*. ACM, 5.
[10] Rosana TV Braga and Paulo Cesar Masiero. 2004. Finding frameworks hot spots in pattern languages. *Journal of Object Technology* 3, 1 (2004), 123–142.
[11] Frank Buschmann, Kelvin Henney, and Douglas Schimdt. 2007. *Pattern-oriented Software Architecture: on patterns and pattern language*. Vol. 5. John wiley & sons.
[12] Nelly Delessy, Eduardo B Fernandez, and Maria M Larrondo-Petrie. 2007. A pattern language for identity management. In *Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on*. IEEE, 7.
[13] Peter Deutsch. 2004. Models and Patterns. In *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons.
[14] Vladan Devedzic. 2002. Software Patterns. In *Handbook of Software Engineering and Knowledge Engineering: Volume II: Emerging Technologies*. World Scientific, 645–671.
[15] Veli-Pekka Eloranta and Marko Leppänen. 2012. Human machine interface patterns for distributed machine control systems. In *Proceedings of the 17th European Conference on Pattern Languages of Programs*. ACM, 3.
[16] Ricardo de Almeida Falbo, Monalessa Perini Barcellos, Julio Cesar Nardi, and Giancarlo Guizzardi. 2013. Organizing ontology design patterns as ontology pattern languages. In *Extended Semantic Web Conference*. Springer, 61–75.
[17] Ricardo de A Falbo, Giancarlo Guizzardi, Aldo Gangemi, and Valentina Presutti. 2013. Ontology patterns: clarifying concepts and terminology. In *Proceedings of the 4th Workshop on Ontology and Semantic Web Patterns*.
[18] Daniela Fogli, Loredana Parasiliti Provenza, and Cristian Bernareggi. 2014. A universal design resource for rich Internet applications based on design patterns. *Universal access in the information society* 13, 2 (2014), 205–226.
[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley* 49, 120 (1995), 11.
[20] Michael Goedicke, Carsten Köllmann, and Uwe Zdun. 2004. Designing runtime variation points in product line architectures: three cases. *Science of Computer Programming* 53, 3 (2004), 353–380.
[21] Dominik Grolimund and Peter Müller. 2006. A Pattern Language for Overlay Networks in Peer-to-Peer Systems.. In *EuroPLoP*. Citeseer, 95–140.
[22] Eduardo Guerra, Felipe Alves, Uirá Kulesza, and Clovis Fernandes. 2013. A reference architecture for organizing the internal structure of metadata-based frameworks. *Journal of Systems and Software* 86, 5 (2013), 1239–1256.
[23] Lilia Gzara, Dominique Rieu, and Michel Tollenaere. 2003. Product information systems engineering: an approach for building product models by reuse of patterns. *Robotics and Computer-Integrated Manufacturing* 19, 3 (2003), 239–261.
[24] Munawar Hafiz, Paul Adamczyk, and Ralph E Johnson. 2012. Growing a pattern language (for security). In *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*. ACM, 139–158.
[25] Christoph Hannebauer, Vincent Wolff-Marting, and Volker Gruhn. 2010. Towards a pattern language for floss development. In *Proceedings of the 17th Conference on Pattern Languages of Programs*. ACM, 15.
[26] Carsten Hentrich. 2009. Synchronization patterns for process-driven and service-oriented architectures. In *Transactions on Pattern Languages of Programming I*. Springer, 103–135.
[27] Carsten Hentrich, Uwe Zdun, Vlatka Hlupic, and Fefie Dotsika. 2015. An approach for pattern mining through grounded theory techniques and its applications to process-driven SOA patterns. In *Proceedings of the 18th European Conference on Pattern Languages of Program*. ACM, 9.
[28] Chin-Yun Hsieh, Yu Chin Cheng, and Chien-Tsun Chen. 2011. Emerging patterns of continuous integration for cross-platform software development. In *Proceedings of the 2nd Asian Conference on Pattern Languages of Programs*. ACM, 9.
[29] Saurabh Hukerikar and Christian Engelmann. 2017. A Pattern Language for High-Performance Computing Resilience. In *Proceedings of the 22nd European Conference on Pattern Languages of Programs*. ACM, 12.
[30] Feldhusen Jörg and Bungert Frederik. 2007. Pattern Languages: An approach to manage archetypal engineering knowledge. *Guidelines for a Decision Support Method Adapted to NPD Processes* (2007).
[31] Douglas Kirk, Marc Roper, and Murray Wood. 2007. Identifying and addressing problems in object-oriented framework reuse. *Empirical Software Engineering* 12, 3 (2007), 243–274.
[32] Barbara Ann Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in software engineering*. Technical Report Version 2.3. EBSE-2007-01.
[33] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.
[34] Stephan Lukosch and Till Schümmer. 2006. Groupware development support with technology patterns. *International Journal of Human-Computer Studies* 64, 7 (2006), 599–610.
[35] Ioanna Lytra, Stefan Sobernig, and Uwe Zdun. 2012. Architectural decision making for service-based platform integration: A qualitative multi-method study. In *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*. IEEE, 111–120.
[36] Michael Mahemoff, Andrew Hussey, and Lorraine Johnston. 2001. Pattern-based reuse of successful designs: usability of safety-critical systems. In *Software Engineering Conference, 2001. Proceedings. 2001 Australian*. IEEE, 31–39.
[37] Jürgen Meister, Ralf Reussner, and Martin Rohde. 2004. Managing product line variability by patterns. In *Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*. Springer, 153–168.
[38] Xiang-xi Meng, Ya-sha Wang, Lei Shi, and Feng-jian Wang. 2007. A process pattern language for agile methods. In *Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific*. IEEE, 374–381.
[39] Daniel Moody. 2009. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering* 35, 6 (2009), 756–779.
[40] Daniel L Moody, Patrick Heymans, and Raimundas Matulevičius. 2010. Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. *Requirements Engineering* 15, 2 (2010), 141–175.
[41] Haralambos Mouratidis, Michael Weiss, and Paolo Giorgini. 2005. Security patterns meet agent oriented software engineering: a complementary solution for developing secure information systems. In *International Conference on Conceptual Modeling*. Springer, 225–240.
[42] Jan de Muijnck-Hughes and Ishbel Duncan. 2012. Thinking towards a pattern language for predicate based encryption crypto-systems. In *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*. IEEE, 27–32.
[43] James Noble. 1998. Towards a pattern language for object oriented design. In *Technology of Object-Oriented Languages, 1998. TOOLS 28. Proceedings*. IEEE, 2–13.
[44] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18.
[45] Ron Porter and Paul Calder. 2003. A pattern-based problem-solving process for novice programmers. In *Proceedings of the fifth Australasian conference on Computing education-Volume 20*. Australian Computer Society, Inc., 231–238.
[46] Fabiano B Ruy, Ricardo A Falbo, Monalessa P Barcellos, Giancarlo Guizzardi, and Glaice KS Quirino. 2015. An ISO-based software process ontology pattern language and its application for harmonizing standards. *ACM SIGAPP Applied Computing Review* 15, 2 (2015), 27–40.
[47] Dina Salah and Amir Zeid. 2009. PLITS: A Pattern Language for Intelligent Tutoring Systems.. In *14th European Conference on Pattern Languages of Programs*.
[48] Douglas C Schmidt, Mohamed Fayad, and Ralph E Johnson. 1996. Software patterns. *Commun. ACM* 39, 10 (1996), 37–39.
[49] Lubor Sesera. 2010. Applying fundamental banking patterns: stories and pattern sequences. In *Proceedings of the 15th European Conference on Pattern Languages of Programs*. ACM, 1.
[50] Ilya Shmorgun, David Lamas, and Eduardo Mercer. 2016. Towards a pattern language for distributed user interfaces. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2712–2718.
[51] Ye Wang, Liping Zhao, Xinyu Wang, Xiaohu Yang, and Sam Supakkul. 2013. PLANT: A pattern language for transforming scenarios into requirements models. *International Journal of Human-Computer Studies* 71, 11 (2013), 1026–1043.
[52] Colin Ware. 2004. *Information Visualization: Perception for Design*. Vol. 2. Morgan Kaufmann.
[53] Danny Weyns. 2009. A pattern language for multi-agent systems. In *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*. IEEE, 191–200.
[54] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering* 11, 1 (2006), 102–107.
[55] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.
[56] Bahman Zamani, Greg Butler, and Sahar Kayhani. 2009. Tool support for pattern selection and use. *Electronic Notes in Theoretical Computer Science* 233 (2009), 127–142.
[57] Uwe Zdun. 2004. Pattern language for the design of aspect languages and aspect composition frameworks. *IEE Proceedings-Software* 151, 2 (2004), 67–83.
[58] Uwe Zdun. 2004. Supporting incremental and experimental software evolution by runtime method transformations. *Science of Computer Programming* 52, 1-3 (2004), 131–163.
[59] Uwe Zdun, Michael Kircher, and Markus Volter. 2004. Remoting patterns: design reuse of distributed object middleware solutions. *IEEE Internet Computing* 8, 6 (2004), 60–68.