Use of patterns for know-how reuse in a model-based systems engineering framework

Quentin Wu, David Gouyon, Eric Levrat, Sophie Boudau

▶ **To cite this version:**

Quentin Wu, David Gouyon, Eric Levrat, Sophie Boudau. Use of patterns for know-how reuse in a model-based systems engineering framework. IEEE Systems Journal, 2020, 14 (4), pp.4765-4776. 10.1109/JSYST.2020.2975116 . hal-02619398

HAL Id: hal-02619398
https://hal.science/hal-02619398

Submitted on 30 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Use of Patterns for Know-how Reuse in a Model-Based Systems Engineering Framework

Quentin Wu, David Gouyon, Éric Levrat, Sophie Boudau

***Abstract*—The increasing complexity of systems being developed requires engineers to review their practices to improve engineering efficiency and meet the needs of a competitive market. To answer these challenges, engineers have always reused their know-how. However, facing today's rising complexity, reuse has to be much more performant. That is why models supported by formal or semiformal languages are preferred to avoid the variability of natural languages interpretation. In this context, Model-Based Systems Engineering (MBSE) made it possible to change the engineering paradigm by proposing a unique, shared system model. To promote and ease MBSE adoption, reuse should be fostered to respect the engineer's working method. A promising method for reusing models is based on the pattern concept. Thus, this paper aims to review and evaluate the pattern concept as a means of transferring know-how and fostering reuse in an MBSE approach.**

***Index Terms*—Model-Based Systems Engineering (MBSE), Pattern, Reuse in Systems Engineering, Systems Modeling**

I. Introduction

Systems tend to incorporate an ever-increasing number of functionalities and operate in a constrained environment.

The design of these increasingly complex systems implies longer engineering phases and greater costs during the design lifecycle of a project. These negative impacts are emphasized by the current document-centered application of Systems Engineering (SE) processes in companies [1]. That is why the SE community has developed Model-Based Systems Engineering (MBSE) [2]. This paradigm promotes a SE methodology that focuses on creating and exploiting system models as the primary means of information exchange between engineers, rather than in documents written in natural language. Popularized by the International Council on Systems Engineering (INCOSE) with the MBSE Initiative[1], it is defined as "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later lifecycle phases". Vogelsang et al. report that its application has been successful in several cases [3] but underlines the fact that companies still struggle with the adoption of MBSE, as many inhibitors remain, such as cultural resistance to change and the lack of support to efficiently implement MBSE (training, maturity of methodologies, clear and committed leadership).

For wider MBSE adoption, several advances seem to be necessary concerning organizational, methodological, and tool perspectives. In particular, from a methodological point of view, it is necessary to construct a means for improving the adoption of MBSE for engineers. As the act of reuse is one of the main enablers of engineering activities, it constitutes a principle that is essential for allowing engineers to use the MBSE methodology. However, the expected benefits include the assumption that the reused modeling artifacts satisfy maturity criteria that guarantee that they have reached a level of quality compatible with the reuse objectives. To formalize reuse activities within an MBSE framework and to benefit from a sufficient maturity level, it appears that the pattern concept is promising, as it is generic and method agnostic [4].

This article focuses on engineering practices that intend to capitalize on know-how by using the pattern concept, especially in an MBSE framework. These approaches propose transforming know-how – which is usually static information (concerning only one individual) – into dynamic information (concerning many individuals) to foster reuse as an enabler of engineering practices. Section II presents an overview of reuse in SE by describing current types of reuse activities by showing associated limitations that justify the reuse of models for efficiently transferring know-how between engineers. As patterns appear to be a promising method for reusing models, section III presents a short history of patterns. Then, section IV reviews the current characteristics, values and limits of patterns for SE, which are also evaluated for MBSE in section V.

II. Background and Context on Know-how Reuse in Engineering Activities

Traditionally, engineers' know-how is constructed from knowledge gained from their experiences [5]. As people become experienced, reuse can be applied faster and automatically to help them to be more efficient in their tasks.

A. Diversity of reuse activities for systems engineering

Reuse – by taking, but not reprocessing, previously used engineering know-how – helps to save time, money, energy and resources. As research has already been conducted for reuse in systems engineering [5]–[13], it appears that there are many

Quentin Wu is with Safran Electrical & Power, Montreuil, 93100, France, and also with the Université de Lorraine, Nancy, 54000, France (e-mail: quentin.wu@safrangroup.com).

David Gouyon is with the Université de Lorraine, Nancy, 54000, France (e-mail: david.gouyon@univ-lorraine.fr).

Éric Levrat is with the Université de Lorraine, Nancy, 54000, France (e-mail: eric.levrat@univ-lorraine.fr).

Sophie Boudau is with Safran Electrical & Power, Montreuil, 93100, France (e-mail: sophie.boudau@safrangroup.com).

[1] http://www.omgwiki.org/MBSE/doku.php (visited on 18/10/2019)

different methods for capitalizing on know-how . These approaches belong to the process of "knowledge transfer" [14] and can be categorized into three different approaches:

- *Opportunistic reuse (OR)*: when the first project was not developed with reusable capacity. It is the lowest incidence of reuse [15] because the act of reuse is the responsibility of the engineer's goodwill;
- *Planned reuse (PR)*: when the first project was developed with reusable capacity. In this case, reuse has been integrated into the development process to highlight and foster reusable contents [16];
- *Variance (V)*: use of something already developed during previous work in a slightly different form. For example, on a product line, there is a common core model, but there are different options [17].

In addition, it is necessary to determine the artifacts that will be reused [18]. Indeed, reuse activities concern:

- *System of Interest (SOI):* this is the system in which the lifecycle is considered;
- *Systems Engineering Activities (SEA*): activities geared towards the development of the SOI.

1) Opportunistic reuse (OR)

"Opportunistic reuse" describes a situation where the "reuse tasks are not performed in any sequence, rather as opportunities arise" [19]. As this approach cannot predict when reuse occurs, it implies that the act of reuse is the responsibility of the engineer's goodwill. This thesis has a considerable implication: in this case, reuse cannot be determined *a priori*, which means that reuse may not occur. It also means that if reuse is performed, benefits are not clear [16].

For example, opportunistic reuse often consists of "copy & paste". Engineers copy a particular type of artifact (e.g., system requirements), which are next adapted to the features of the current project under development [20]. In the software community, it corresponds to an equivalent action, which consists of "extending software with functionality from a third-party software supplier that wasn't originally intended to be integrated and reused" [21]. Even though the authors present this approach as an enabler to rapid development, they acknowledge the fact that the quality of the product, when released, is not as high as with other reuse methods. In the same way, the authors of [22] used this approach on abandoned projects and called it "scrapheap software development". They also acknowledged the fact that identifying reusable assets is the "burden" of the developer and that no safeguards exist to inform them that they might reuse content inappropriately.

The authors in [23] investigated the reasons behind these issues and described different levels of knowledge from the point of view of an engineer to underline the cognitive issues that occur during reuse (Fig. 1). This description explains the reasons that can encourage engineers to start a reuse process, or on the contrary, to prevent them from starting one. Indeed, in the entire information space (global), each individual has his/her own information space (local). Inside each individual information space, different layers exist and correspond to graduation from what is well known (L1) to what is only expected to exist (L3) and worse, what is expected to exist but actually does not (L4-L3). In this way, it is more likely that engineers will reuse information included in L1 than in L3 since risks are managed, as they are better understood.

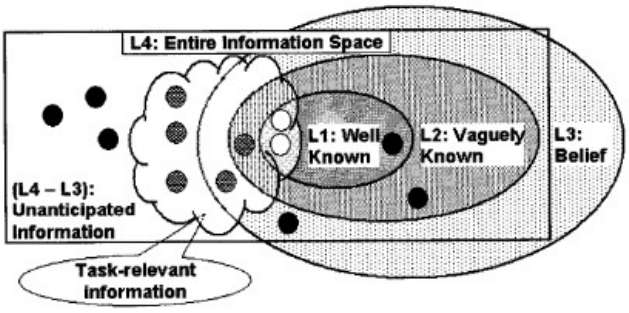

Fig. 1 Different levels of knowledge about a high-functionality application (HFA) extracted from [23]

An opportunistic approach might help for rapid development, but it does not ensure good quality reuse or that performing reuse will be viable in the future. As presented, the information space of each stakeholder of the project has to be considered to ensure mature, viable and efficient reuse, which is expected to achieve clear benefits.

2) Planned reuse (PR)

As presented in the previous subsection, engineering can benefit from reuse, but to be efficient, it needs to be planned as it requires proper documentation and design [24]. It is, therefore, important to ensure that future development cycles incorporate a reuse approach that is planned and formalized. The "planned reuse" approach is expected to provide a greater and clearer benefit than the "opportunistic approach" since risks are managed upstream. Several research works have studied and developed processes compliant with a "planned reuse" approach. Whether on the SOI, SEA or both, the goal of each work was to provide a framework for reuse that is well defined and that guides the engineers in their work.

One of the trends is to shift the engineering paradigm from "problem solving" towards "decision making". Currently, many development teams are spending a long time solving problems from one "point" solution to another one that is more mature and closer to customer needs ("point-based design") [25]. The authors of [26] emphasize that an inversion of the paradigm is needed, and from their point of view, it is feasible to transfer the workload towards the "front" phases of engineering activities. Indeed, by accumulating knowledge upstream, it is possible for the development team to define a set of possible designs (Fig. 2). As a result, "set-based design" aims to reduce rework and improve decision making by maturation before making key decisions.

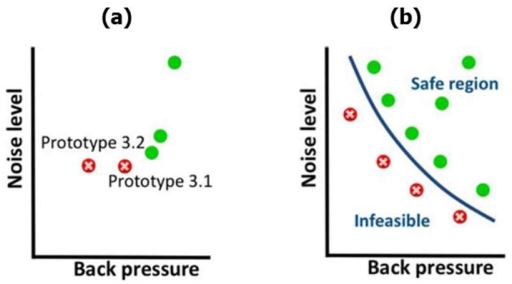

Fig. 2 (a) Point-based design, (b) Set-based design extracted from [26]

Thus, it seems necessary to breakdown the problem into subproblems where it is possible to make decisions easier. The commercial off-the-shelf (COTS) approach provides this capability, as it is a "divide and conquer" design paradigm that consists of breaking down a problem into solvable subproblems by already existing components. Consequently, the authors of [27] noted that COTS may be selected and implemented for technical (shorter development time), business and organizational (reduce overall system development costs), or strategic reasons (access a technology that cannot be developed internally). However, in systemic thinking, the "whole" is greater than the sum of its "parts". For this reason, the advantages of COTS are accompanied by integration issues, earlier identified by the authors of [28], which are functionality and performance (what it is expected to do), interoperability (no standard exists), product evolution (risk of no longer meeting the need) and vendor behavior (false promises). Beyond these concerns, it appears that the key to using COTS is the need for an efficient selection method, which understands companies' needs and situations for improving integration.

Research also targets processes where reuse can take place to define reuse activities. These activities can, therefore, take place throughout the development cycle. For example, the authors of [14] started in the innovation process. Over the six case studies they analyzed, the authors identified a "six-stage reuse-for-innovation process". Their study emphasizes the need for engineers to look for other's works and evaluate them to create situations where they are more likely to innovate. Thus, organizational structures need to adapt to allow innovative engineers to be aware of both traditional and nontraditional approaches to reuse and create new roles that become part of a community that fosters reuse inside organizations. The work conducted by the authors of [29] involved an organizational change. Indeed, they proposed adapting the requirement engineering process by creating a local reuse library for each engineer and a shared reuse library managed by a reuse manager (RM). Many examples are also available in the software community [30], such as the PABRE framework [31]. The authors propose the use of software requirements patterns (SRP) as a means to capture and reuse requirements knowledge validated by experts. To produce software requirement specification of better quality, their framework is composed of a metamodel of SRP, a method of reuse through SRP, a catalog of 111 SRP, and a software system that supports the management and use of the catalog. The pattern concept allows showing the different forms to achieve and serves as a basis to improve the set of requirements.

As previously mentioned, reuse activities can take place throughout the development cycle. It is possible, for example, to mention the work on COSYSMO [32], an extension of the constructive cost model (COCOMO) [33]. Indeed, the latter has been extended to include the reuse capacity legacy from older systems by defining reuse categories and weights for each of the categories [13].

Planned reuse requires significant effort in addition to the constraints of a project. However, it appears that greater benefits are expected compared to opportunistic reuse in terms of quality, time development and communication between engineers.

3) Variance (V)

Beyond opportunistic and planned reuse, another method for reusing artifacts from previous works occurs through the concept of “variability”, which can be defined as the “capacity of a characteristic to vary” [20]. Therefore, these variabilities define the perimeter of the reuse domain. This concept appears to be adapted towards product line engineering, which does not concern only the system but also the global context in which several aspects matter, such as products, processes, enterprise management and organization. In the context of reuse in engineering activities, it is necessary to detail products and processes. Indeed, products of a product line are made of a generic product (the result of a capitalization process) and their applications (the result of a reuse process). Thus, it appears that it is necessary to define processes for capitalization and reuse to develop applications from a generic product and vice versa. This implies the need to model variability, which can be done using MBSE modeling methods, which are currently advancing in systems engineering.

In that sense, many research works appear to leverage modeling techniques for variance [34] to model the scope of architecture options or, in other words, the modeling of product variabilities. Indeed, on the one hand, it is necessary to capture the model of the system (requirements, functions, etc.) and, on the other hand, it is necessary to capture system characteristics (common, variant), to represent the dependency between artifacts (constraints, variabilities) and to determine an achievable combination of elements [20]. That is why modeling techniques are relevant to reducing complexity due to the high number of variabilities and dependencies, to make explicit achievable combinations, to ease impact analysis and to capitalize on already explored combinations to avoid “reinventing the wheel”. To leverage these modeling techniques, several languages have been proposed, such as FORE (family-oriented requirements engineering), FODA (feature-oriented domain analysis), OVM (orthogonal variability models), and CVL (common variability language) [35], [36].

Variance embeds the advantages of previous reuse approaches (standardization, time saving, efficiency, and quality improvement) but also eases the trade-off analysis [37] by allowing the analysis of design alternatives [38]. However, challenges still remain concerning product lines: implementation (maturity, cultural change, impact on lifecycle), objectives (answer to a need, ROI is expected, optimization of the industrial process), dependency between

systems engineering processes (configuration management, capitalization and reuse, modeling).

Whereas reuse is a relatively well-documented practice in software and manufacturing, the formalization of this practice in the systems engineering domain is relatively new, and little has been reported on actual industrial applications [39].

B. The necessary evolution of the method for reusing know-how

Different approaches to reuse are presented (OR, PR, V). In an SE context, it is necessary to link these approaches to the different levels of complexity that can be encountered when transferring know-how for reuse. These levels are not clear-cut categories and are defined by the capability to allow efficient reuse provided by means of sharing used (Table I).

TABLE I
SYSTEM COMPLEXITY LEVELS AND MEANS OF SHARING (MOS)

MoS \ Levels of complexity			Level 1	Level 2	Level 3
Oral	Raw	Discussion Interview Drawings	+	-	- -
	Reuse	Presentation Lecture	++	~	-
Text	Raw	Rationale Comment Email	+	+	-
	Reuse	Manual Book Technical Document	++	+	~
Model	Raw	Individual (or specific) Model	++	++	+
	Reuse	**Model Reuse?**	++	++	++

Efficiency: - - Very low; - Low; ~ Correct; + High; ++ Very high

For each method of sharing, there are two kinds of usage. On the one hand, there is raw usage (white background) where the use of know-how in a specific context is only possible once. On the other hand, there is reuse usage (gray background) where the intention is to transmit know-how more formally to enable reuse in a different context. The question mark on the last line of Table I shows that even if modeling techniques are being deployed, work still must be done to formalize a reuse approach for models to increase engineering efficiency.

1) Level 1: "low" complexity

At this level of complexity, the artifact transmission is oral, and the transfer is usually performed through discussions, interviews, drawings, or returns of experiences. However, oral transfer is not sufficient for transferring a large amount of know-how, as it takes time and implies a strong involvement of both instructors and apprentices.

2) Level 2: "intermediate" complexity

Contrary to oral transmission, the advantage of text and documents is that it remains and does not leave with people when they retire so that the human is not the weak link. Moreover, once a document is written, the involvement of instructors decreases. However, problems can occur concerning the variability of natural language interpretation and document size explosion.

3) Level 3: "high" complexity

As complexity continues to increase, the use of models is preferable, as models are more expressive and less ambiguous than textual documents [40]. Indeed, at this level of complexity, the size and number of documents require a considerable amount of work to ensure information consistency across engineering teams and throughout the various iterations of the system lifecycle [41]. This is why disciplines such as MBSE are emerging, where the model forms the heart of all the systems engineering activities and is the basis of many of the project artifacts to ensure consistency [42]. Indeed, architectural elements or time-dependent dynamic behavioral simulations allow a more complete examination and early exploration of the logical and behavioral characteristics of the architecture [43].

Assuming that complexity is increasing over time and will continue to increase – as well as data that engineers need to manage – it implies that a new method for transferring know-how needs to be defined and developed. If one considers level 3 of the system's complexity (Table I), this means that engineers cannot answer complexity only through oral and text and that they have to change their method of working by using models.

C. Towards the reuse of models to transfer know-how between engineers

Human-centric information created by engineers is essential but has the detrimental effect of being "*static*" as it remains implicit and biased (engineer's point of view). Indeed, information is fixed in one engineer's mind, making it difficult to transfer to someone else to foster reuse [44], [45]. "*Dynamic*" information among engineering teams is a critical challenge for many companies that need to manage complex systems, as information must be shared, comprehensive, and coherent among the projects [46]. This aspect is very important because it allows a better comprehension of the SOI and SEA. For example, in requirement engineering [29], at least 55% of engineers use "copy & paste" to reuse requirements or groups of requirements, and 50% duplicate the full specification.

These practices have little added value, as they do not convey a strong understanding of the system, especially concerning its behavior. This is why there is a need to promote an efficient method for transferring know-how to facilitate its circulation and reuse, and this is the reason that the current expectations are to promote models rather than documents written in natural language because of its variability of understanding. Indeed, the format of the transfer has a prominent role (Table I). As complex systems produce a considerable amount of information and require efficient descriptions or specifications, reuse can help to counter today's rising complexity [47].

The next step for improving engineering efficiency is to focus on reusing know-how through models. One method that looks particularly promising is through the adoption of patterns by proposing generic guides to ease and systematize the construction of complex systems [48]. As they can be used in

all stages of the development cycle [10], reuse of patterns seems to be the most suitable form of reuse for complex [49] systems engineering on the condition that this concept can be formalized in a modeling process.

III. A Little History of Patterns

Most people in the pattern community attribute the first promoter of the value of the "pattern" to Alexander and his coauthors in a book on architecture, urban design and community livability [50]. The book formalizes a language called a "pattern language", composed of a myriad of patterns that helped the authors express design in terms of relationships between the parts of a house and the rules for transforming these relationships [51]. They began to identify patterns with the idea that “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice” [50]. The same way engineers reuse their knowledge based on their previous experience, the author of [52] notes that the authors of [50] "did not invent these patterns, they came from observation and testing; and only then were they documented as patterns".

The pattern approach has been introduced in various engineering fields, such as software, requirements, telecommunications and control systems engineering [53]. The authors of [54] were the first to propose object-oriented patterns in the software community when engineers started to apply the concepts developed by Alexander and his coauthors. The goal was to improve quality and to facilitate code writing by adopting good practices. The authors of [55], also known as the "Gang of Four", described 23 software design patterns such as composite, iterator, and command. A design pattern is a general, reusable solution to a recurring problem in the design of object-oriented applications; it describes a proven solution for solving software architecture problems. Design patterns are not a finished design (concrete algorithm) but a structured description of computer programming. This means that they are independent of programming languages. The use of design patterns offers many advantages. First, it solves a design problem due to a proven solution validated by experts. Therefore, design speed and quality increase while costs decrease. In addition, since design patterns are widely documented and known to many developers, they also facilitate communication because the awareness of design patterns by software engineers makes it unnecessary to go into detail during the design (UML diagram). Design patterns have been widely accepted and encouraged in other software domains for writing patterns to capture their experience. However, facing the number of pattern collections, the authors of [56] proposed to reduce the catalog of software patterns to reduce confusion and promote cleaner reuse. They defined the syntax and semantics of patterns for detecting and analyzing redundancies. Their work led them to range patterns from identical patterns into lower degrees of similarity, resulting in a smaller collection of patterns.

In the field of SE, the value of patterns appears essential towards the growing complexity of systems and the difficulty of capturing a large body of knowledge. That is why Barter, the author of [6], proposed the creation of a systems engineering pattern language. This language is a collection of patterns that, when combined, address problems larger than the problems that an individual pattern can address. This language can be graphically represented by a pattern map, and Barter described the generic elements that a "minimal" pattern should contain. The author of [6] captured and managed the systems engineering body of knowledge (SEBOK) information by using patterns and pattern languages and proposed using concept maps to represent relationships between individual patterns in a pattern language.

Pursuing this direction, the author of [57] also proposed the use of SE patterns to capture the information in the SEBOK. However, unlike the author of [6], who considered a pattern language as a collection of patterns, without mention of relationship types between patterns, the author of [11] detailed the relationships between patterns at different levels and emphasized the need for a working group to describe these links. Other works have used the pattern concept in systems engineering, especially in the product information system field, where the authors of [10] proposed a methodological framework based on the reuse of patterns throughout the system lifecycle. The authors of [58] proposed pattern libraries to support a methodological framework for the conception of product information systems.

The growing interest within the SE community towards MBSE [2] and specific characteristics of patterns (generic, various application method) led the INCOSE to start an "MBSE Patterns Challenge Team" in 2013 as a part of the INCOSE MBSE Initiative. In 2016, it became the "INCOSE MBSE Patterns Working Group"[2], which aims to "advance the availability and awareness of practices and resources associated with the impactful creation, application, and continuous improvement of MBSE patterns over multiple system life cycles".

After this short review of the history of patterns, a focus on patterns in systems engineering is made in the next section to understand the fundamental concepts needed to foster reuse in systems engineering.

IV. Patterns for Systems Engineering

As recalled in the previous section, the pattern concept and its applicability in the field of systems engineering have already been discussed. However, this concept has many facets, making it difficult for a beginner to identify the target and the means to use patterns. Although many characteristics of patterns are recurrent in various research works, it requires synthesis work. The purpose of this section is to improve the comprehension of what is a pattern in systems engineering.

A. Recurrent characteristics of patterns

The study of previous research allows the identification of recurrent characteristics of patterns. They are synthesized in Table II and detailed in this subsection.

[2] http://www.omgwiki.org/MBSE/doku.php?id=mbse:patterns:patterns (visited on 18/10/2019)

TABLE II
RECURRENT CHARACTERISTICS OF PATTERNS IN THE LITERATURE

RC1.	A pattern does not appear out of the blue; patterns are "mined" from the know-how.
RC2.	A pattern is represented in a static format that owns specific attributes, one being the existence of a triptych of three elements {Context, Problem, Solution}.
RC3.	A pattern objective is to efficiently transfer information between stakeholders at the right level of abstraction to foster reuse.
RC4.	The applicability of a pattern is determined by certain conditions, such as their perimeter and performance of the application, but also their maturity level.

Sometimes, similar designs are made entirely independently by different engineers [59]. This phenomenon acknowledges the fact that the same design elements exist in multiple designs, and the study and documentation of such designs foster reuse among projects. Indeed, it prevents "reinventing the wheel" and provides a vocabulary for the design concepts that a project can share. This is consistent with the notion that patterns "are not created from a blank page; they are *mined*" [60]. It appears that systems engineering patterns are embedded in existing designs and that it is necessary to find a mechanism for identifying them. These patterns are called "buried patterns" by the authors of [61] and represent a scientific obstacle, as the mining techniques differ depending on whether the design is represented with formal or nonformal languages. As the process of "mining" appears to be essential for creating pattern languages, various approaches have been identified to write patterns from mined elements. For example, the author of [62] was interested in three processes of mining: mining by interviewing, mining by teaching pattern writing, and mining by borrowing (from the literature).

However, based on Delano's classification [63], it is possible to classify mining processes into three categories:

- *Individual contributions*: Writers of the pattern use their own experiences or the experiences of their colleagues. The contribution is based on their ability to write good patterns.
- *Second-hand contributions*: Patterns are written based on interviews with experts or by guiding another person in the writing of patterns. However, it can also come from borrowing patterns from the literature or from companies in the same domain that can share information.
- *Workshops/meeting contributions*: As in Haskins [62] during an INCOSE tutorial, this approach consists of grouping participants by groups of approximately ten people to brainstorm the elements of a pattern, and a moderator and facilitator can be included to keep the discussion on track.

When mining a pattern, the next question is the format in which it is captured. Despite the differences between formats depending on the language used (textual or modeling), it appears that a minimal set of information is always provided, consisting of interdisciplinary generic attributes that characterize a pattern. In addition to an evocative name that constitutes its identification property, a pattern seems to possess an inherent triptych composed of {context, problem, solution}. The latter is mentioned by many research works [6], [64]. The authors of [56] defined a "minimal triangle", where the triptych defines the core meaning of a pattern (Fig. 3).

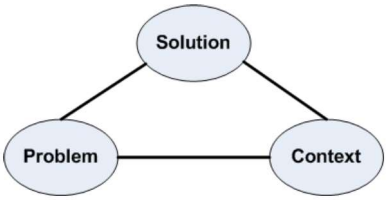

Fig. 3 The minimal triangle extracted from [56]

The minimal triangle summarizes the idea that a pattern provides a solution to a recurring problem in a particular context. Thus, if one element is missing, this pattern is called a "trivial pattern". However, a general consensus enlarges the minimal elements needed in a pattern, such as the generic pattern with minimal elements that the authors of [6] described: an evocative name, the triptych {context, problem, solution}, the forces (or tensions) that influence the application of a pattern, and related patterns.

In different examples given by the authors of [11], [53], [61], and [64], elements from the generic pattern appear or are at least written in the text of the article. However, they can also be broken down into smaller elements, such as the formalism of design patterns described by the authors of [55], which refers to a solution through different elements: application, structure, advantages, code sample or utilizations (known uses). The authors of [53] conducted a survey that allowed them to list a recommended systems pattern form (Fig. 4) and demonstrate that "systems engineers and architects are most interested in the rationale for the pattern followed by an example of how to apply the pattern and known uses of the pattern".

Form Heading	Explanation
Pattern Name:	The name of the pattern should be descriptive to enable the pattern user to understand the usage.
Aliases:	Other names by which the pattern may be known
Keywords:	Keywords which assist in locating appropriate patterns in a repository
Problem Context:	Brief discussion of the types of situations in which the problem may occur - it should be broad enough to allow for any number of situations in which the problem may arise
Problem Description:	What is the problem this pattern can be used to solve?
Forces:	What challenges exist in the problem being addressed by the pattern, and the problems in applying the pattern? May also include constraints the pattern may impose if used. May describe the pattern from multiple views
Pattern Solution:	Discussion on how the pattern solves the problem being addressed.
Diagrams:	This can be one or more diagrams necessary to represent the pattern. This can be any notational method desired.
Interfaces:	Discussion of the critical interfaces or information flows necessary in implementing the pattern - what parameters of the interface can change and which ones can not. What are the interface dependencies, if any?
Resulting Context:	What are the unaddressed issues remaining when the pattern is applied/used.
Example:	An example of how the pattern may be applied. Usually in the form of a diagram or model
Pattern Rationale:	Why the pattern works
Known Uses:	Where else is the pattern being used in other places or applications?
Related Patterns:	Other patterns that may work in conjunction or in association with this pattern
References:	Other information that may be useful in understanding or applying the pattern
Author(s):	Who documented the pattern? May add a date if desired.

Fig. 4 Recommended systems pattern form extracted from [53]

The authors of [53] also emphasize the fact that concepts used in systems engineering represent higher levels of complexity and abstraction than the prevailing notions of Alexander in architecture. For instance, the architecture of the underlying

concepts of control command requires a more complex notation than the sketch (Fig. 5) used by Alexander in [50].

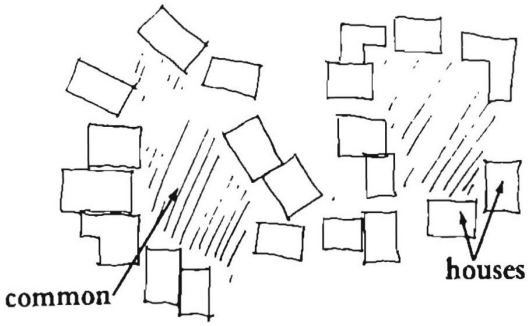

Fig. 5. Pattern 37: House clusters extracted from [50]

Thus, the authors of [61] used the enhanced functional flow block diagram (eFFBD) to represent the model of their control command (Fig. 6). Such pattern descriptions rely on formal conceptual foundations in the form of a metamodel, which can be used for the management, application, and cataloging of patterns specific to the field of systems engineering. Ultimately, patterns are methodologically agnostic, so the most important task is to know what to capture due to the mining process.

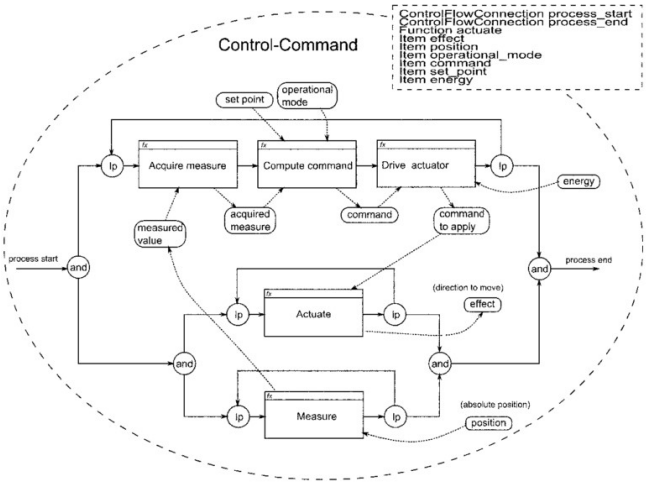

Fig. 6 A systems engineering pattern extracted from [61]

B. *Value of patterns in SE*

Therefore, these recurrent characteristics are not sufficient for defining what a pattern is. Indeed, they miss an aspect that is the value of patterns. As written in the introduction, the fundamental idea behind the idea of "reuse" is the capacity to set static motion information to make it dynamic. This way, information has to be available to become an artifact that engineers want to use. Like models, patterns are abstractions or a set of abstractions of reality and not a magical solution. They allow people to solve complex problems by leveraging experience and know-how from their predecessors. The results of a study conducted in the open source software community by the author of [65] determined whether the 23 design patterns introduced by the authors of [55] were used to document changes in the code. The authors of [65] observed that 11.4% of projects with one developer use patterns, approximately 20% for projects with 2-9 developers, and 61.5% when the team size exceeds 10 people (Fig. 7).

		Team size						
		1	2	3	4	5-9	10+	Total
Developer	0	226	107	43	26	29	5	436
using	1	29	21	8	6	6	2	72
patterns	2		2		1	2	3	8
	3						1	1
	4						1	1
	10						1	1
Total		255	130	51	33	37	13	519
Observed projects with patterns		11.4%	17.7%	15.7%	21.2%	21.6%	61.5%	16.0%
Estimated projects with patterns[a]		11.4%	21.5%	30.4%	38.3%	50.8%	82.2%	18.4%

[a] Using team size=1 to estimate propability of a developer using patterns

Fig. 7 Use of patterns depending on the team size extracted from [65]

These results seem to imply that explicit documentation of a used pattern allows efficient communication in a development team. Indeed, one of the main arguments in favor of using patterns in SE is the capacity to improve communication because with the current complexity of systems; it is almost impossible for systems engineers to envision all the details of a system. Thus, the capacity of patterns to deliver at each level of the development the correct amount of information will allow its quick adoption and most importantly its active use, as the author of [65] concluded in his study: "design patterns are adopted for documenting changes and thus for communication in practice by many of the most active open source developers".

In terms of communication, patterns, therefore, offer the possibility to create a common lexicon between systems architects. This will help foster a common understanding of systems architecture, validated by experts. This enables the spread of common design features for reuse in different projects, reducing cost and time to design a new system. However, as patterns aim at converting static information into dynamic information, the patterns themselves should not be static. There is a critical need for patterns to evolve to become mature. In this way, patterns will foster reuse and help to control the complexity of a system.

As stated in the introduction concerning know-how, reuse is fundamental for allowing engineers to increase their efficiency. Thus, there should be patterns that address:

- The SOI by reusing sets of requirements, architectural elements, etc., as illustrated for the architecture of a Joint Emergency Services Control Center [66]
- SEA reuses methodological guidelines recursively at various system levels, as illustrated in the system of innovation patterns [8].

C. *Limits of patterns in SE*

Nevertheless, patterns are not "silver bullets", and some limitations may be expressed. One of the criticisms is the difficulty in creating and innovating by applying only the reuse process. This criticism considers that it will be impossible to make breakthroughs ("future") by combining patterns ("past"). However, this is not the only objection, as the author of [67] described, other reasons to avoid patterns can be listed:

- When addressing new or unique requirements that have not been solved before;

- When the requirements require a unique solution, e.g., aesthetics over function;
- When the pace of technological change does not warrant the use of patterns.

Of course, the strict reuse of information by copy/paste cannot allow innovation or emulate creativity. Thus, it is necessary to find another way to achieve innovation through reuse. The authors of [14] partially addressed this question by mining a process pattern (Fig. 8) based on case studies from the Jet Propulsion Laboratory. They proposed a six-stage process for innovation in which the key action consists of searching and evaluating others' works to exploit existing ideas and generate new combinations. In this context, the perimeter should not be restricted to the search for solutions to the current know-how base, as the intended behavior is not about replicating but integrating recombined ideas. This pattern may not be necessary and sufficient, but it shows how patterns can help to formalize know-how to foster innovation during the next projects. Patterns also aim to reduce the engineering phase length, reduce costs and provide more free time to innovate.

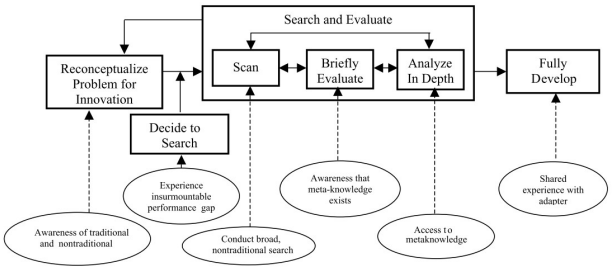

Fig. 8 Knowledge reuse process for innovation extracted from [14]

The second objection emphasizes the fact that patterns should not be static but dynamic. As it has been written in many places in this article, patterns, in order to be used, "should offer the same comfortable learning experience as a conversation with trusted colleagues" [11]. This means that there is an organizational challenge for motivating engineers to contribute to a pattern database: experts have to understand the importance of performing documentation and validation of patterns to train junior systems engineers. This also means that support must be provided to enable engineering teams to find their bearings. One can imagine, for example, that the pattern database could take the form of a library, a concept that can be easily understood and customized according to needs.

The third objection considers that patterns cannot evolve as quickly as new technologies and are, therefore, quickly obsolete. However, if this objection is relevant for low-level patterns that address recurrent technological problems, more abstract patterns – concerning high-level architecture, for example – can remain accurate. Moreover, in long-term project industries, such as aerospace, it appears that if senior engineers know-how is not documented as patterns, the capacity of recognizing a problem and applying a solution is no longer available for junior engineers that have to relearn standard designs, resulting in higher costs for developing a new system [67]. Furthermore, this objection is mainly focused on the SOI, as it does not consider SEA, which can be maintained over time inside each company, independently from the evolution of the market.

As the interest in MBSE increases, it is important to also examine the work performed to integrate the pattern concept in this framework. The integration of the OMG System Modeling Language (OMG SysML) and its consequences on how to define problems and describe solutions are particularly interesting and will be examined in the next section.

V. Patterns for Model-Based Systems Engineering

Although research has been conducted to assess whether the pattern concept can be applied in the systems engineering field, such as [11], [52], and [61], the value of patterns in an MBSE framework has not been fully explored. However, it appears crucial to consider all the different needs, requirements and constraints of the different stakeholders in the early design stages. Perceived by many companies as a time loss, it appears that introducing reuse capacity in MBSE methodologies allows the design of a new project with much less human effort, benefiting from the reuse of the already existing system models [68]. The identification and capture of patterns in the existing system models extend and increase reusability [69], which benefits new projects by accelerating identical engineering phases by quickly injecting these patterns into system models during the design of a specific solution.

The purpose of this section is to show the usefulness of the capitalization and reuse of system models by showing how the pattern concept can be implemented in MBSE and thus favors its adoption at a larger scale. First, presentations are made in this section on the value of reuse and patterns in an MBSE framework. Then, in a second step, the limits of the reuse with patterns in MBSE are analyzed.

A. Value of reuse in an MBSE framework

Models are abstractions or a set of abstractions of reality (i.e., the reality can be represented under different consistent views), which means that it can be easy to reuse a model in a new project since there are no physical limitations. Moreover, models can also support decision making by presenting needed information [70].

However, the act of reuse requires capturing previous know-how in an explicit format. However, depending on the type or the set of modeling artifacts that will be reused, obstacles such as the complexity of the system under design or the heterogeneity of methodologies and tools appear. Indeed, reusing existing modeling artifacts (even if their designs have been made to be reusable) is harder than expected. As the author of [12] states, the "biggest problem is to transfer and manage the knowledge [of] what is actually available for reuse". The emphasis is on the fact that it is necessary for systems engineers to be aware of system and engineering assets that can be defined and propagated among teams designing complex systems. The author proposed extending the software-oriented Object Management Group Reusable Asset Specification (OMG RAS) standard definition to address SE assets in the OMG System Modeling Language (OMG SysML). The definition of these assets can use either a top-down or a bottom-up approach. However, the creation of asset libraries is not enough, as the purpose is to allow engineers to reuse these assets in their

ongoing projects. The author of [12] emphasizes the fact that users should have the ability to search, publish, and reuse assets in defined libraries and catalogs, without any specific technical prerequisite. However, the update of assets is only handled through notifications within a tool and does not address the maturity scale concerning asset reuse. Contrary to the previous work [12], the authors of [71] do not focus on the creation of assets but propose an approach concerning the adaptation of promising reusable assets during a model reuse process, especially on the adaptation of OMG Unified Modeling Language (OMG UML) activity diagrams to new use cases, in the context of web engineering. This work proposes semiautomatically creating an activity diagram from existing activity diagrams according to the input use case diagram. It aims to help engineers specify functional requirements by reusing models of existing web applications. Even though this approach is not presented in an MBSE framework, the fact that, between OMG UML and OMG SysML, use case diagrams are identical and that activity diagrams are close allows considering a transposition in the SE field. In the case of variant modeling in MBSE, the numerous perspectives and architectural levels induce exponential information growth as well as greater complexity due to the extra dimensionality of variants [72]. The authors of [73] proposed an approach for building and exploiting composable architectures applied to the design and development of a product line of complex systems in the aerospace and defense market. By leveraging reference architectures that are inherently reusable, the authors sought four benefits: aligning the team, starting fast, aligning the resultant systems, and learning. They chose OMG SysML as the core language to define descriptive models of the composable system reference architecture and extended it to define parametric models. This methodology allowed the product line to evolve more readily as the propagation of information by adding, updating or modifying new components was done automatically. However, they observed that the "initial learning curve was steep for the team". To ease adoption of such methodologies, work is needed, whether it is in the development of tools, interfaces or in the concepts used. As the previous work concerns the physical layer, the authors of [72] focused their attention on the development of a functional architecture that can adapt according to changes made in the logical layer of a system of systems (SoS). Indeed, in these systems, it appears that the physical layer is constrained by the fact that the interacting systems already exist. The results of their study are an MBSE process that consists of the integration of a system model before the consideration of the variants. It requires that the system model should contain both the original configuration and the variant configuration. This representation is necessary in case old technologies have not been abandoned. They also investigate the aspects of including variant modeling into the OMG SysML, with a focus on extending an existing and operating model to support a new variant in the case where a similar technology is used. This tendency to reuse models in an MBSE framework is growing, as proved by the current request for proposal for the future version 2 of SysML that seeks to provide the capacity to foster reuse of models [74].

B. Value of patterns in an MBSE framework

The introduction of a reuse capacity in MBSE frameworks has proven to improve engineering efficiency in engineers' work. However, there is still a steep learning curve for organizations to adopt MBSE methodologies. Therefore, it is necessary to help engineers to "quickly identify not only valid architectural solutions but optimal value solutions for the mission need" [73]. Thus, it appears that the pattern concept could be an answer to this challenge. Indeed, work has been performed to introduce patterns during various engineering phases of the development cycles. For example, the author of [75] describes behavioral construct patterns (Fig. 9) to facilitate the modeling of the behavior of the system. Instead of thinking at the level of atomic graphical elements, these constructs defined a structured method for representing elementary constructs of behavior. This approach is very useful for structuring the thinking of engineers, and it also makes it possible to overcome a big problem in modeling tools, which is called the "connect policy" by the author of [75], which refers to the aesthetic balance of diagrams that is irretrievably broken when new elements are added. In this way, he advocated the use of an "insert policy", such as in the construction of the functional flow block diagram (FFBD), where the resizing of the diagram is automatic when new elements are inserted. The proposed behavioral construct patterns will allow engineers to work in an algorithmic way of thinking, which implies a higher modeling level to focus more on the expected behavior than on the aesthetics of the diagrams.

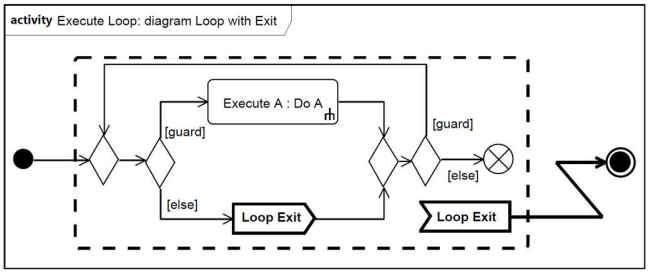

Fig. 9 Loop exit construct extracted from [75]

To help engineers focus on what is important, patterns should guide development to avoid deviation. For example, the authors of [76] proposed a process for the development of mechatronic systems based on a SysML design pattern. Their intent was to demonstrate that adequate guidelines for modeling benefit the development process by allowing traceability of all information within the system model. This approach proves to be particularly helpful for facilitating impact analysis in later lifecycle phases and for reuse for future projects. Indeed, being able to trace a change from the stakeholder's requirement to the component's requirement helps engineers in the allocation of an adequate physical part, and it also facilitates the integration of a model in a new project at the same level of abstraction. Further, helping engineers in their decision making can be crucial, which is why the authors in [66] explore architecture patterns to make trade-off analyses on SoS and provide guidelines for the reuse of existing architectures in the development of future architectures. Today, the analysis of large-scale SoS is not easy to undertake. However, with the use

of patterns, expressed as OMG SysML models, this analysis became possible even if all system details were not available. The mining of patterns that represent the SOI can help to create a model of the SoS and thus induce a better understanding of its behavior. This approach permits changes in the system to optimize a particular aspect of the SoS while ensuring that the overall system is not compromised or that undesirable emergent behaviors result.

Completing the work of [11] on patterns, the author of [77] proposes an engineering paradigm where patterns are reusable models that enables what he calls pattern-based systems engineering (PBSE). With the advent of MBSE, this modeling framework has led to patterns that can be configured or specialized into product lines or into product systems by following the definition of the MBSE pattern working group (see Section III). In this context, the authors of [78] developed their own approach, as they see "patterns as reusable models" and applied them to requirements and design. At a high level, they constitute a generic system pattern model that can be customized for the needs, configuration, or use of an enterprise so that engineers can benefit from the concepts of MBSE without being an expert in modeling methodologies. The previous approach has been applied in the V&V stage [8] and in the pharmaceutical market [79].

C. *Limits of patterns in an MBSE framework*

Patterns in an MBSE framework have the same limitations as in SE. However, since they are expressed graphically, it is harder to hide intellectual property without losing consistency and understanding, such as in a customer-supplier relationship. It is one of the capacities that will be crucial to foster MBSE adoption and, thus, the use of patterns for model reuse.

MBSE patterns also have specific limitations due to model reuse. On the one hand, the interoperability between tools is currently not mature, which means that it is very difficult to transfer a model in a tool different from the one in which it was made. On the other hand, currently available modeling languages are not yet well defined in their syntax and semantics. This has led many users to make choices and customize these languages to meet their needs. This generated a strong divergence in how to express patterns, thus reducing the capability to reuse them as is. However, some research has shown how the gap can be reduced by introducing more formalization inside the SE design process via the definition of an ontology to formalize concepts [80]. In accordance with this, the current request for proposal for the future version 2 of SysML (mentioned previously in section V.A.) also wants to converge the different concepts towards a common understanding.

Finally, whether for models or patterns, efficient reuse can only occur when a sufficient level of maturity has been reached. This level must provide the user with a sufficient level of confidence to reuse a pattern without error and ambiguity. A first step is to evaluate the maturity of such capitalized patterns, as done in the automated production systems domain by the authors of [81] on the maturity on control modules in libraries, or by the authors of [82], who define model maturity metrics to improve the ability to assess a model's maturity for systems engineering technical review (SETR). A second step is to improve the general maturity of reuse approaches, as done in the software domain by the authors of [83], who use metrics inspired by the capability maturity model.

All of the above limitations are obstacles towards the creation of an MBSE framework that will foster the use of patterns as a guide for the development of complex systems. Once those limitations are resolved in methodological terms, work will still be necessary to implement these capabilities in a tool and make them "user friendly" for end users.

VI. CONCLUSION & PERSPECTIVES

The introduction emphasized the fact that shortened engineering cycle periods are the main issue for systems engineers. This is the reason why system modeling is becoming an increasingly important part of any systems engineering project [42]. For a wider adoption of MBSE, in addition to implementation in a software tool, this paper highlighted the strong methodological need to capitalize on previous projects to reuse know-how for new projects. For this purpose, this article reviews current practices of reuse, focusing on the pattern concept, to support the transition towards MBSE. This concept offers the possibility to make information dynamic between stakeholders during the development of complex systems and to share it and foster its reuse for future projects. Thus, several research works have been reported in the design and application of reusing know-how within an MBSE approach. Despite limitations concerning the management of intellectual property or the interoperability of models, the adoption of MBSE due to patterns was discussed, as patterns appear to be a key element for allowing efficient systems engineering.

From a methodological perspective, the pattern concept allows us to abstract elementary constructs. This approach helps to structure engineers' way of thinking. In other words, it will help engineers focus on what is important by guiding future developments (guidelines for the reuse of existing architecture) while benefiting from MBSE methodologies without being an expert. However, it is first necessary to evaluate the maturity of models as well as the maturity of the pattern reuse process. Thus, our future work will also aim at developing maturity scales that will provide maturity metrics for both SOI and SEA. Indeed, it will then be possible to assess the amount of effort required to improve one's level of maturity. Our ongoing work also aims to formalize an MBSE methodology for developing complex systems by leveraging patterns (identification, extraction) [84] and thus improving the maturity level of the pattern reuse process. As patterns need to be mined, it appears that the act of capitalization (by abstracting/extracting patterns) is not self-evident and implies the ability to make know-how explicit. Similarly, the reuse of patterns involves adapting the prescribed solution to a given context. Since these processes will surely be performed manually at first, the next step to improving engineering effectiveness concerns the development and adoption of MBSE software tools that integrate capitalization, selection, reuse, and update capacities for patterns in the form of libraries, for example. The latter will make it possible to constitute a body of know-how that will be shared among engineers.

REFERENCES

[1] E. R. Carroll and R. J. Malins, "Systematic Literature Review : How is Model- Based Systems Engineering Justified ?," Sandia Nat. Lab., Albuquerque, NM, USA, Tech Rep., 2016.

[2] J. A. Estefan, "Survey of Model-Based Systems Engineering (MBSE) Methodologies," *INCOSE MBSE Initiat.*, 2008.

[3] A. Vogelsang, T. Amorim, F. Pudlitz, P. Gersing, and J. Philipps, "Should I Stay or Should I Go? On Forces that Drive and Prevent MBSE Adoption in the Embedded Systems Industry," 2017.

[4] Q. Wu, D. Gouyon, P. Hubert, and É. Levrat, "Towards Model-Based Systems Engineering (MBSE) Patterns To Efficiently Reuse Know-How," *Insight*, vol. 20, no. 4, pp. 31–33, 2017.

[5] L. A. Bollinger and R. Evins, "Facilitating model reuse and integration in an Urban Energy Simulation platform," *Procedia Comput. Sci.*, vol. 51, no. 1, pp. 2127–2136, 2015.

[6] R. H. Barter, "A Systems Engineering Pattern Language," in *INCOSE International Symposium*, 1998, pp. 350–353.

[7] R. J. Cloutier, "Model Driven Architecture for Systems Engineering," *INCOSE Int. Work.*, no. September, 2008.

[8] D. Cook and W. Schindel, "Utilizing Mbse Patterns To Accelerate System Verification," *Insight*, vol. 20, no. 1, pp. 32–41, 2017.

[9] N. Gautam, R. B. Chinnam, and N. Singh, "Design reuse framework : a perspective for lean development," *Int. J. Prod. Dev.*, vol. 4, no. 5, pp. 485–507, 2007.

[10] L. Gzara, D. Rieu, and M. Tollenaere, "Product information systems engineering: An approach for building product models by reuse of patterns," *Robot. Comput. Integr. Manuf.*, vol. 19, no. 3, pp. 239–261, 2003.

[11] C. Haskins, "Application of patterns and pattern languages to systems engineering," in *INCOSE International Symposium*, 2005, pp. 1619–1627.

[12] A. Korff, "Re-using sysml system architectures," in *Proceedings of the 4th International Conference on Complex Systems Design and Management*, 2013, pp. 257–266.

[13] G. Wang, R. Valerdi, and J. Fortune, "Reuse in systems engineering," *IEEE Syst. J.*, vol. 4, no. 3, pp. 376–384, 2010.

[14] A. Majchrzak, L. P. Cooper, and O. E. Neece, "Knowledge Reuse for Innovation," *Manage. Sci.*, vol. 50, no. 2, pp. 174–188, 2004.

[15] A. Rockley, P. Kostur, and S. Manning, *Managing Enterprise Content: A Unified Content Strategy*. New Riders, 2003.

[16] D. C. Rine and R. M. Sonnemann, "Investments in reusable software. A study of software reuse investment success factors," *J. Syst. Softw.*, vol. 41, no. 1, pp. 17–32, 1998.

[17] N. Niu, J. Savolainen, Z. Niu, M. Jin, and J.-R. C. Cheng, "A Systems Approach to Product Line Requirements Reuse," *IEEE Syst. J.*, vol. PP, no. 99, pp. 1–10, 2013.

[18] F. Pfister, V. Chapurlat, M. Huchard, C. Nebut, and J.-L. Wippler, "A Proposed Meta-Model for Formalizing Systems Engineering Knowledged, Based on Functional Architecture Patterns," *Syst. Eng.*, vol. 15, no. 3, 2012.

[19] A. Sen, "The Role of Opportunism in the Software Design Reuse Process," *IEEE Trans. Softw. Eng.*, vol. 23, no. 7, pp. 418–436, 1997.

[20] A. Le Put, *Systems Product Line Engineering Handbook*, AFIS. Éditions Cépaduès, 2016.

[21] S. Jansen, S. Brinkkemper, I. Hunink, and C. Demir, "Pragmatic and Opportunistic Reuse in Innovative Start-up Companies," *IEEE Softw.*, vol. 25, no. 6, pp. 42–49, 2008.

[22] G. Kotonya, S. Lock, and J. Mariani, "Scrapheap Software Development: Lessons from an Experiment on Opportunistic Reuse," *IEEE Softw.*, vol. 28, no. 2, pp. 68–74, 2010.

[23] Y. Ye and G. Fischer, "Supporting reuse by delivering task-relevant and personalized information," *Proc. 24th Int. Conf. Softw. Eng. - ICSE '02*, p. 513, 2002.

[24] S. Younoussi and O. Roudies, "Capability and maturity model for Reuse: A comparative study," *2016 2nd Int. Conf. Cloud Comput. Technol. Appl.*, pp. 302–308, 2016.

[25] D. Sobek II, A. C. Ward, K. Liker, Jeffrey, D. Sobek, A. C. Ward, and K. Liker, Jeffrey, "Toyota s Principles of Set-Based Concurrent Engineering," *Sloan Manage. Rev.*, no. Winter, pp. 67–83, 1999.

[26] B. M. Kennedy *et al.*, "Reducing Rework by Applying Set-Based Practices Early in the Systems Engineering Process," *Syst. Eng.*, vol. 17, no. 3, pp. 278–296, 2014.

[27] J. Hedman and B. Andersson, "Selection method for COTS systems," *Procedia Technol.*, vol. 16, pp. 301–309, 2014.

[28] B. W. Boehm and C. Abts, "COTS integration: plug and pray?," *IEEE Comput.*, vol. 32, no. 1, pp. 135–138, 1999.

[29] R. Darimont, W. Zhao, C. Ponsard, and A. Michot, "Deploying a Template and Pattern Library for Improved Reuse of Requirements Across Projects," *Proc. - 2017 IEEE 25th Int. Requir. Eng. Conf. RE 2017*, pp. 456–457, 2017.

[30] M. Irshad, K. Petersen, and S. Poulding, "A systematic literature review of software requirements reuse approaches," *Inf. Softw. Technol.*, vol. 93, no. September 2017, pp. 223–245, 2018.

[31] C. Palomares, C. Quer, and X. Franch, "Requirements Reuse with the PABRE Framework," *Requirements Engineering Magazine*, 2014.

[32] J. Fortune and R. Valerdi, "Considerations for Successful Reuse in Systems Engineering," *AIAA Sp.*, pp. 1–8, 2008.

[33] B. W. Boehm, "Software Engineering Economics," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 1, pp. 4–21, 1984.

[34] J. Ryan, S. Sarkani, and T. Mazzuchi, "Leveraging Variability Modeling Techniques for Architecture Trade Studies and Analysis," *Syst. Eng.*, vol. 17, no. 1, pp. 10–25, Mar. 2014.

[35] E. K. Budiardjo and E. M. Zamzami, "Feature Modeling and Variability Modeling Syntactic Notation Comparison and Mapping," *J. Comput. Commun.*, vol. 2, no. 02, p. 101, 2014.

[36] S. Nurcan, P. Soffer, M. Bajec, and J. Eder, *Advanced Information Systems Engineering: 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings*. Springer International Publishing, 2016.

[37] T. Weilkiens, *Variant Modeling with SysML*. Tim Weilkiens, 2012.

[38] R. J. Malak, L. Tucker, and C. J. J. Paredis, "Composing Tradeoff Models for Multi-Attribute System-Level Decision Making," no. May, pp. 993–1005, 2009.

[39] H. G. Chalé Góngora, M. Ferrogalini, and C. Moreau, "How to Boost Product Line Engineering with MBSE – A Case Study of a Rolling Stock Product Line," in *Proceedings of the 5th International Conference on Complex Systems Design and Management*, 2014, pp. 239–256.

[40] F. Mhenni, N. Nguyen, and J.-Y. Choley, "SafeSysE: A Safety Analysis Integration in Systems Engineering Approach," *IEEE Syst. J.*, vol. 12, no. 1, pp. 1–12, 2016.

[41] R. S. Kalawsky *et al.*, "Bridging the gaps in a model-based system engineering workflow by encompassing hardware-in-the-loop simulation," *IEEE Syst. J.*, vol. 7, no. 4, pp. 593–605, 2013.

[42] J. Holt, S. Perry, R. Payne, J. Bryans, S. Hallerstede, and F. O. Hansen, "A Model-Based Approach for Requirements Engineering for Systems of Systems," *IEEE Syst. J.*, vol. 9, no. 1, pp. 252–262, 2015.

[43] B. Ge, K. W. Hipel, K. Yang, and Y. Chen, "A novel executable modeling approach for system-of-systems architecture," *IEEE Syst. J.*, vol. 8, no. 1, pp. 4–13, 2014.

[44] D. Mourtzis, M. Doukas, and C. Giannoulis, "An Inference-based Knowledge Reuse Framework for Historical Product and Production Information Retrieval," *Procedia CIRP*, vol. 41, pp. 472–477, 2016.

[45] P. Demian and R. Fruchter, "An ethnographic study of design knowledge reuse in the architecture, engineering, and construction industry," *Res. Eng. Des.*, vol. 16, no. 4, pp. 184–195, 2006.

[46] A. Ben Miled, "Reusing knowledge based on ontology and organizational model," *Procedia Comput. Sci.*, vol. 35, no. C, pp. 766–775, 2014.

[47] C. Trummer *et al.*, "Searching extended IP-XACT components for SoC design based on requirements similarity," *IEEE Syst. J.*, vol. 5, no. 1, pp. 70–79, 2011.

[48] T. Cochard, "Contribution à la génération de séquences pour la conduite de systèmes complexes critiques," Université de Lorraine, 2017.

[49] D. May and P. Taylor, "Knowledge Management with patterns," *Communications of the ACM*, vol. 46, no. 7, pp. 94–99, 2003.

[50] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language*. 1977.

[51] J. O. Coplien, "Idioms and patterns as architectural literature," *IEEE Softw.*, vol. 14, no. 1, pp. 36–42, 1997.

[52] R. J. Cloutier, "Applicability of patterns to architecting complex systems," Stevens Institute of Technology, 2006.

[53] R. J. Cloutier and D. Verma, "Applying the concept of patterns to systems architecture," *Syst. Eng.*, vol. 10, no. 2, pp. 138–154, 2007.

[54] K. Beck and W. Cunningham, "Using Pattern Languages for Object-Oriented Programs," *OOPSLA-87 workshop on the Specification and Design for Object-Oriented Programming*, 1987. .

[55] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[56] A. Gaffar and N. Moha, "Semantics of a Pattern System," *Proc. STEP Int. Work. Des. Pattern Theory Pract. IWDPTP05*, 2005.

[57] C. Haskins, "1.1.2 Using Patterns to Share Best Results - A proposal to codify the SEBOK," *INCOSE Int. Symp.*, vol. 13, no. 1, pp. 15–23, 2003.

[58] A. Conte, M. Fredj, I. Hassine, J.-P. Giraudin, and D. Rieu, "A Tool and a Formalism to Design and Apply Patterns," in *Object-Oriented Information Systems*, 2002, pp. 135–146.

[59] A. Gaffar and N. Moha, "Semantics of a Pattern System," *Proc. STEP Int. Work. Des. Pattern Theory Pract. IWDPTP05*, 2005.

[60] R. S. Hanmer and K. F. Kocan, "Documenting architectures with patterns," *Bell Labs Tech. J.*, vol. 9, no. 1, pp. 143–163, 2004.

[61] F. Pfister, V. Chapurlat, M. Huchard, C. Nebut, and J.-L. Wippler, "A Proposed Meta-Model for Formalizing Systems Engineering Knowledge, Based on Functional Architecture Patterns," *Syst. Eng.*, vol. 15, no. 3, 2012.

[62] C. Haskins, "Using Patterns to Transition Systems Engineering from a Technological to Social Context," *Syst. Eng.*, 2007.

[63] D. E. DeLano, "Patterns mining," in *The Pattern Handbook: Techniques, Strategies, and Applications*, 1998, pp. 87–96.

[64] C. Cauvet, D. Rieu, B. Espinasse, J.-P. Giraudin, and M. Tollenaere, "Ingénierie des systèmes d'information produit : une approche méthodologique centrée réutilisation de patrons," *Inforsid*, pp. 71–90, 1998.

[65] M. Hahsler, "A quantitative study of the adoption of design patterns by open source software developers," in *Free/Open Source Software Development*, Igi Global, 2005, pp. 103–124.

[66] R. S. Kalawsky, D. Joannou, Y. Tian, and A. Fayoumi, "Using architecture patterns to architect and analyze systems of systems," *Procedia Comput. Sci.*, vol. 16, no. March 2015, pp. 283–292, 2013.

[67] R. J. Cloutier, "Toward the Application of Patterns to Systems Engineering," *Syst. Eng.*, no. January 2005, pp. 73–80, 2005.

[68] U. Shani and H. Broodney, "Reuse in model-based systems engineering," *9th Annu. IEEE Int. Syst. Conf. SysCon 2015 - Proc.*, pp. 77–83, 2015.

[69] S. C. Spangelo *et al.*, "Applying model based systems engineering (MBSE) to a standard CubeSat," *IEEE Aerosp. Conf. Proc.*, pp. 1–20, 2012.

[70] M. Russell, "Using MBSE to Enhance System Design Decision Making," *Procedia Comput. Sci.*, vol. 8, pp. 188–193, 2012.

[71] S. Paydar and M. Kahani, "A semi-automated approach to adapt activity diagrams for new use cases," *Inf. Softw. Technol.*, vol. 57, no. 1, pp. 543–570, 2015.

[72] M. Di Maio, G. D. Kapos, N. Klusmann, and C. Allen, "Challenges in the modelling of SoS design alternatives with MBSE," *2016 11th Syst. Syst. Eng. Conf. SoSE 2016*, 2016.

[73] C. Oster, M. Kaiser, J. Kruse, J. Wade, and R. Cloutier, "Applying Composable Architectures to the Design and Development of a Product Line of Complex Systems," *Syst. Eng.*, vol. 19, no. 6, pp. 522–534, Nov. 2016.

[74] OMG, "Systems Modeling Language (SysML®) v2 Request For Proposal (RFP)," 2017.

[75] L. Gasser, "Structuring activity diagrams," in *14th IFAC Symposium on Information Control Problems in Manufacturing, Bucharest, Romania*, 2012.

[76] G. Barbieri, K. Kernschmidt, C. Fantuzzi, and B. Vogel-Heuser, "A SysML based design pattern for the high-level development of mechatronic systems to enhance re-usability," in *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 2014, vol. 19, no. 3, pp. 3431–3437.

[77] W. Schindel, "Requirements Statements Are Transfer Functions: An Insight from Model-Based Systems Engineering," in *INCOSE International Symposium*, 2005, pp. 1604–1618.

[78] W. Schindel and T. Peterson, "Introduction to Pattern-Based Systems Engineering (PBSE): Leveraging MBSE Techniques," *INCOSE Int. Symp.*, vol. 23, no. 1, p. 1639, 2013.

[79] J. L. Bradley, M. T. Hughes, and W. Schindel, "Optimizing delivery of global pharmaceutical packaging solutions, using systems engineering patterns," *20th Annu. Int. Symp. Int. Counc. Syst. Eng. INCOSE 2010*, vol. 3, pp. 2441–2447, 2010.

[80] H. Chalé, O. Taofifenua, T. Gaudré, A. Topa, N. Lévy, and J.-L. Boulanger, "Reducing the Gap Between Formal and Informal Worlds in Automotive Safety-Critical Systems," *INCOSE Int. Symp.*, vol. 21, no. 1, pp. 1306–1320, 2011.

[81] B. Vogel-Heuser, J. Fischer, E.-M. Neumann, and S. Diehm, "Key maturity indicators for module libraries for PLC-based control software in the domain of automated Production Systems," in *16th IFAC Symposium on Information Control Problems in Manufacturing*, 2018.

[82] J. D. Gaskell and C. N. Harrison, "Improved System Engineering Technical Review's Entrance/Exit Criteria with Model Maturity Metrics," in *5th IEEE International Symposium on Systems Engineering*, 2019.

[83] L. V. Manzoni and R. T. Price, "Identifying extensions required by RUP (Rational Unified Process) to comply with CMM (Capability Maturity Model) levels 2 and 3," *IEEE Trans. Softw. Eng.*, vol. 29, no. 2, pp. 181–192, 2003.

[84] Q. Wu, S. Boudau, D. Gouyon, and É. Levrat, "Capitalization and reuse with patterns in a Model- Based Systems Engineering (MBSE) framework," in *5th IEEE International Symposium on Systems Engineering*, 2019.