Best Practices for Domain-Specific Modeling. A Systematic Mapping Study

Gerald Czech, Michael Moser, Josef Pichler
Software Competence Center Hagenberg GmbH, Hagenberg, Austria
Email: (gerald.czech, michael.moser, josef.pichler)@scch.at

***Abstract*—Model-driven software development comes in different styles. While standard-based approaches leverage existing language standards (e.g. UML), tooling, and even development processes, domain-specific modeling (DSM) requires domain-specific languages and tool support to be created prior to the actual software development. The design, implementation, and testing of languages and tool support require a wide spectrum of methods and techniques where each of them brings also additional complexity and challenges. To tackle these DSM-specific challenges, best practices have been collected from various application domains and published in literature to guide the development and application of DSM solutions. This work explores existing best practices by conducting a systematic mapping study. We identify and classify studies reporting practical guidance on domain-specific modeling and present best practices from literature. Moreover, we discuss how best practices overlap, complement, or contradict each other. From a total of 309 best practices in 19 papers, we compiled 189 unique best practices. The systematic and comprehensive compilation of best practices is intended to facilitate industrial adoption of DSM in various domains.**

***Keywords*-domain-specific modeling; best practices; systematic mapping study;**

I. Introduction

Domain-specific modeling (DSM) [1] is a model-driven approach for software development. DSM proposes to model a software system in a high-level language and to fully generate code from these high-level models. The goal is to raise the level of abstraction beyond programming by specifying the solution in a language that directly uses concepts and rules from a specific application domain. The final product is generated next in a chosen programming language from these high-level specifications. DSM has several advantages that are suitable to tackle complexity in the development of today's software systems. The direct use of concepts and rules from a specific application domain closes the gap between the application domain and the solution domain. In this way, DSM can increase the productivity [1], ease the maintenance [2], and increase the reusability [3].

DSM as characterized above does not only replace classical software technologies but also emphasizes the seamless methodology from domain concepts down to generated code instructions. The therefore required spectrum of methods and techniques includes aspects of domain-driven design [4] down to advanced techniques from compiler construction [5]. Of course, this wide range of individual techniques and methods also gives you the opportunity to make mistakes. These mistakes can not only cancel out the potential benefits of DSM, but they can also lead to the failure of entire development projects. Best practices for domain-specific modeling may help software developers to master the complexity of DSM and to counteract the numerous risks during adoption of DSM.

In software development, a best practice is a well-defined method that contributes to a successful step in product development. Throughout the software industry, several best practices are widely followed. Some of the more commonly used are: an iterative development process, requirement management, quality control, and change control. With the recent industrial adoption of DSM we also find specific DSM best practices in literature compiled from various industrial projects. The goal of this paper is thus to systematically collect these best practices and to structure them to get a comprehensive overview. Therefore, we conduct a systematic mapping study (SMS) [6] which is described in Section II together with specific research questions we intend to answer. Section III presents the results of the conducted systematic mapping study. The results of the SMS are discussed in Section IV including the compact compilation of 189 best practices collected from 19 publications.

II. Systematic Mapping Study

Systematic literature reviews (SLR) [7] have been widely adopted by the software engineering community over recent years. A form of SLR, that intends to map the available research, rather than answering specific questions, are systematic mapping studies (SMS) [8]. This systematic mapping study essentially follows the process described in [6]. Steps of their systematic mapping process include definition of research questions, conducting the search for relevant papers, the screening of search results using inclusion and exclusion criteria, keywording of abstracts to create a classification schema, and data extraction and mapping.

A. Research Questions

The overall goal of our study is to reveal and pinpoint work providing practical guidance and advice on the development of domain-specific modeling solutions. This interest is framed by the following research questions:

- *RQ-1: Where and how are practices on domain-specific modeling reported?* This question aims at identifying reports providing guidance and advice on how to best

create domain-specific modeling solutions. Moreover, we want to classify research type, style, and form of presentation of best practices.

- *RQ-2: From which sources are practices derived?* This question aims to identify and categorize the sources the presented studies draw their practices from.
- *RQ-3: What activities and aspects of DSM are covered by practices?* The research question answers if studies focus on specific aspects and activities of DSM development only. Moreover, we want to reflect if certain activities got greater coverage than others.
- *RQ-4: Which practices are reported and how do they interrelate?* This question aims to provide a compact but nevertheless comprehensive compilation of practices in the field of domain-specific modeling and their interrelations. To answer this question normalization and matching of practices reported by different studies is required.

B. Conducting Search

The search for relevant studies comprised the identification of base literature, definition of search terms, searching digital libraries, snowballing on selected studies and complementing search results with results from search engines and meta-search engines. First, we identified base literature which we consider prototypical for our case, applied snowballing and used this preliminary set of result literature to elaborate on initial search terms. In general, the chosen search terms reflected a combination of the field of research (i.e. domain-specific modeling) and terms matching best practices.

To cover the field of domain-specific modeling we expanded it to the field of domain-specific languages. For both terms we defined abbreviations, i.e. DSM, domain-specific modeling, DSL, and domain-specific language. To include best practices in our search results we defined the synonyms, i.e. guidelines, principles, rules, lessons learned, success factors, requirements, experiences and patterns. Moreover, we expanded these terms with opposite terms, i.e. anti-pattern, worst practices and pitfalls. We iteratively expanded and refined search terms on the basis of the quality and number of search results yielded.

For this systematic mapping study we searched the digital libraries *ACM*, *IEEExplore*, *Springer Link*, and *Science Direct*. The search was conducted in January 2018. Table I shows the final set of search queries we used. The applied query is a Boolean AND operation where one operand matches any synonym for *DSM* and the second operand matches any synonym, or opposite, for best practice. Search was limited to titles of studies only. Quality and number of results were satisfying for *ACM* (52 results), *IEEExplore* (41 results), and *Science Direct* (6 results). However, due to limited search options, searching *Springer Link* yielded unspecific and a high number of unrelated results. Nevertheless, to document results we queried *Springer Link* via *Google Scholar* using a source filter for *Springer Link* (25 results).

Table I
SEARCH QUERIES USED IN DIGITAL LIBRARY SEARCH.

Library	Search Terms and Query
ACM	acmdlTitle:(+("best practices" principles rules requirements guidelines experiences "lessons learned" "success factors" patterns "worst practices") +(DSL "domain specific language" "domain-specific language" DSM "domain-specific modeling" "domain specific modeling"))
IEEE	(Document Title:best practices OR Document Title:principles OR Document Title:rules OR Document Title:requirements OR Document Title:guidelines OR Document Title:experiences OR Document Title:lessons learned OR Document Title:success factors OR Document Title:patterns OR Document Title:worst practices) AND (Document Title:Domain specific language OR Document Title:DSL OR Document Title:Domain-specific languages OR Document Title:Domain-specific modeling Document Title:Domain specific modeling OR Document Title:DSM)
Science Direct	Title("domain specific language" OR "domain-specific language" OR DSL OR "domain specific modeling" OR "domain-specific modeling") AND Title("best practice" OR "principles" OR "rules" OR "requirements" OR "guidelines" OR "experiences" OR "lessons learned" OR "patterns" OR "worst practices")
Springer	TITLE(("best practice" OR "principles" OR "rules" OR "requirements" OR "guidelines" OR "experiences" OR "lessons learned" OR "patterns" OR "worst practices") AND (DSM OR "domain-specific modeling" OR "domain specific modeling" DSL OR "domain-specific language" OR "domain specific language"))

Moreover, we supplemented results from digital library search with search results from meta-search engines, i.e. Google Scholar (34 results). Finally, we added base literature we identified during an initial web-search to the final set of search results. After eliminating duplicates from the results our search produced 143 studies in total.

C. Paper Screening

We screened search results for inclusion in our study on the basis of inclusion and exclusion criteria. Defined criteria are:

- *Inclusion criteria:* books, peer reviewed studies published in journals, conferences, and workshops reporting formalized, or semi-formalized practices or reflecting on experiences and lessons learned from domain-specific modeling or domain-specific languages.
- *Exclusion criteria:* Paper is outside of the software engineering domain. Tutorials, slides, gray literature, keynotes, and proceeding summaries. Paper does not refer to domain-specific modeling or domain-specific languages. Reports on internal DSLs [9], as they are constrained on a host language and thus are not as relevant for DSM. Reports which do not share experiences, patterns, lessons learned or practices.

The first and second author of this study screened title, abstracts and keywords of the 143 studies and evaluated studies for inclusion. This produced 16 conflicting assessments, which were resolved by an independent evaluation of the third author. Reasons for exclusion of papers are: no experiences, lessons learned, guidelines, etc. are reported (68), paper is outside the software engineering domain (34), reports on internal DSLs (5), reports on new approaches rather than on experiences (7) and others. The high number of papers which do not report best practices is due to the appearance of a search term in a different context, as for example in *A Textual Domain Specific Language for Requirement Modeling* [10]. The high number of papers outside of the domain is due to the usage of the terms *DSL* and *DSM* as abbreviations in other research domains, such as medicine or power engineering. The screening process produced 19 studies which we included in our systematic mapping study. Finally, we applied snowballing to supplement our final list of papers with additional yet undiscovered studies. In the context of this work only backward snowballing, i.e. identification of studies listed as references in a primary study, was feasible. However, no additional study was identified.

D. Classification of Relevant Papers

To create a classification schema we applied the following strategy. First, we followed the recommendation of Peterson et al. [6] to extract keywords from paper abstracts and to iteratively update a classification schema. Second, we enriched the created classification schema with keywords derived from research questions. Finally, we added a generic paper classification schema as suggested in [6]. Table II shows the basic dimensions of the created schema.

Table II
DIMENSIONS OF CLASSIFICATION SCHEMA OF SMS.

Dimension	Description
Research Type Facet	Assignment of papers according to research type facet proposed by [6]: validation research, evaluation research, solution proposal, philosophical papers, opinion papers, experience papers.
Formalism	Level of formality of the presentation of best practices, i.e. formal, semi-formal, or anecdotal.
Sources	The sources from which selected studies derive best practices, e.g. industrial projects.
Form	Categorization of form of practice, i.e. best/worst practice, lessons learned, guideline, or pattern.
Notation	The form of DSL notation, i.e. textual, graphical, or both, a paper refers to.
User	Categorization of users of DSM-solutions as presented in examples of the selected studies.
Software Life-Cycle	Assignment of studies to one or many software life-cycle phases, i.e. requirement gathering and analysis, design, implementation, test, deployment, and maintenance of the DSM solution.

E. Data Extraction and Mapping of Studies

During data extraction tow main tasks were performed. First, we conducted the classification of studies according to the created schema and second, we extracted and compiled best practices reported in the studies. To classify papers according to our schema we again screened abstracts and scanned content of all included studies and extracted data for each dimension. Again the first two authors of this study participated in the data extraction. Conflicting views and ambiguities were discussed among all authors.

To extract and compile best practices from the selected studies, screening and scanning of papers was not sufficient. For each study we extracted practices contained and assigned a short name, a description, and the addressed topic to the best practice. Name, description, and topic were extracted as given in the study. If data was missing, e.g. missing short name or no explicit topic assignment, we extracted the data from the descriptions of best practices. After extraction of best practices from each study, we merged all best practices into a single compilation. This required normalization and abstraction of best practices. Obviously, naming and description of best practices could vary between studies, even though the same practice contributing to the same topic is meant. Maintaining traceability between best practices from a single study and best practices of the merged compilation was crucial. Moreover, we learned that extracting a precise short name and topic in the first place, helped to ease the mapping process. The second and third author of this paper provided a mapping of best practices. Conflicting assessments were discussed and resolved among all authors.

III. RESULTS

Systematic search and screening of results resulted in 19 studies. Fig. 1 shows the number of papers per year. Without

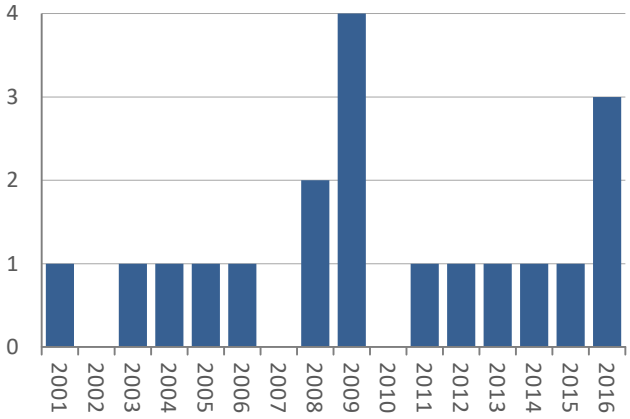

Figure 1. Selected publications per year.

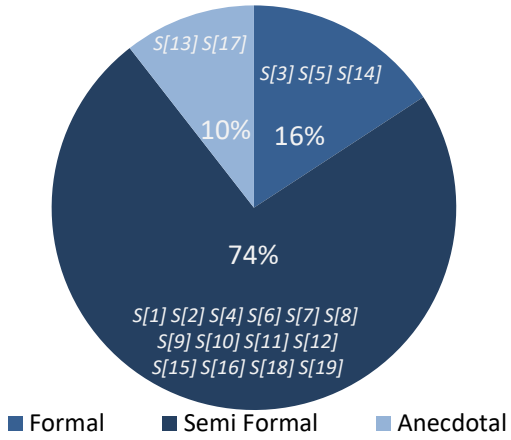

Figure 2. Style (formal vs. informal) of report.

Table III
PUBLICATIONS PER RESEARCH TYPE FACET.

Research Type Facets	Primary Studies
Validation research	none
Evaluation research	[S2]
Solution proposal	none
Philosophical paper	[S3] [S4] [S5] [S6]
Opinion paper	none
Experience paper	[S7] [S8] [S9] [S10] [S11] [S12] [S13] [S14] [S15] [S1] [S16] [S17] [S18] [S19]

restricting our query with respect to publication date, the selected papers range from 2001 to 2016. Fig. 2 depicts the distribution of formalisms used to report best practices. The style how best practices are reported ranges from anecdotal experience description (2) to formal descriptions using pattern languages (3). Most studies (14), however, use a semi-formal style which use basic structuring of best practices. Besides the formalism, we identified five different forms of reporting practices, as shown in Fig. 3. 4 studies use the term *best practices* literally, one the opposing term *worst practices*, and the remaining studies refer to *lessons learned* (6), *patterns* (5), and *guidelines* (3). To ease readability of this paper, we used the term *best practice* for all forms, except otherwise needed. Furthermore, Table III shows the classification of research type facets following Wieringa et al.'s [11] definition which is also recommended for SMS by Petersen [6]. Unsurprisingly with respect to the focus of our SMS, far the most publications are *experience papers* (14) besides *philosophical papers* (4) and *evaluation research* (1). Non of the selected studies qualify as *validation research*, *solution proposal*, or *opinion paper*. Table IV associates our primary studies with origins mentioned in the publications. If no source is give we classified them as *Author's Experience*. Far the most publications term *Industrial Projects* as source of best practices.

Moreover, we picked two central topics of domain-specific modeling to further classify primary studies: first, the *notation* of a DSL and second, the *users*. In Table V we list whether best practices address textual notation, graphical notation, or both. In Table VI we list DSM user-types recognized by best practices, i.e. software developers and business users. In Table VII we associate publications and phases of the software development life cycle [12]. Fig. 4 maps the phases in Table VII with research types in Table III. Our results are concluded by Fig. 5 showing the number of best practices for each publication in chronological order.

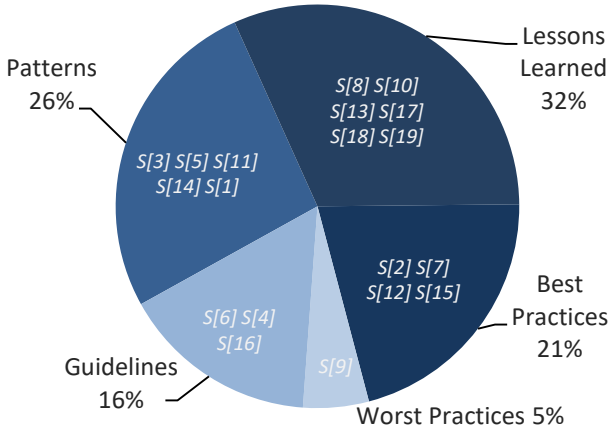

Figure 3. Form of practices reports.

Table IV
SOURCES OF REPORTED BEST PRACTICES.

Source of best practices	Primary Studies
Industrial Projects	[S7] [S9] [S10] [S11] [S13] [S14] [S1] [S17] [S18] [S19]
DSL Tool Development	[S7] [S12]
Literature Study	[S3] [S4] [S5] [S6] [S7]
Author's Experience	[S2] [S8] [S15] [S16]

IV. DISCUSSION

A. Research Question 1: Primary Studies

The initial motive of our research was to identify studies which report best practices on domain-specific modeling (*RQ-1*). Studies published prior 2001 were not detected

Table V
NOTATION OF DSL.

Notation	Primary Studies
Textual	[S4] [S6] [S11] [S12] [S15] [S18] [S19]
Graphical	[S13] [S14] [S1] [S16] [S17]
Both	[S2] [S3] [S5] [S7] [S8] [S9] [S10]

Table VI
USERS OF DSL.

DSL User	Primary Studies
Developer and Architect	[S2] [S6] [S7] [S12] [S1] [S16] [S19]
Business user	[S9] [S10] [S11] [S18]
Both	[S3] [S4] [S5] [S8] [S13] [S14] [S15] [S17]

Table VII
COVERED PHASES IN SOFTWARE DEVELOPMENT LIFE CYCLE.

Phase	Primary Studies
Req. Gath. & Analysis	[S3] [S4] [S7] [S8] [S9] [S10] [S13] [S16] [S17]
Design	[S2] [S3] [S4] [S5] [S6] [S7] [S8] [S9] [S10] [S11] [S13] [S15] [S1] [S16] [S17] [S18]
Implementation	[S3] [S4] [S5] [S7] [S8] [S10] [S11] [S12] [S13] [S14] [S15] [S1] [S17] [S18]
Testing	[S8] [S13] [S15] [S19]
Deployment	[S10] [S14]
Maintenance	[S13] [S15] [S1]

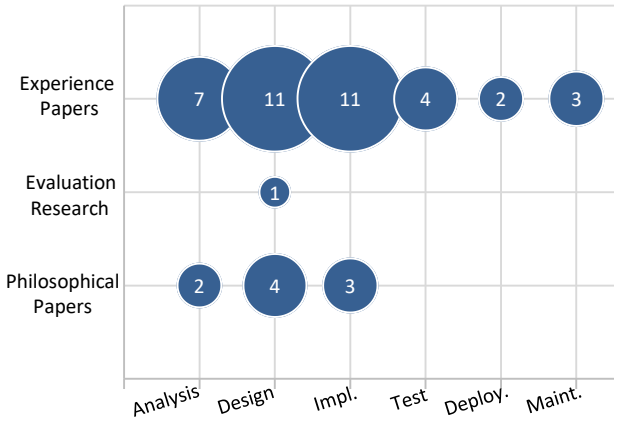

Figure 4. Research Type Facets vs. Software Development Phases

by our search mainly due to the search terms *domain-specific modeling* and *domain-specific language*. We considered synonyms for DSL, like *little languages*, *task-specific languages*, or *specific languages* as out of scope. Similarly, more generic terms like *modeling* or *programming language*, which would have delivered results prior 2001, are as well considered out of scope for this study. With the exception of 2007 and 2010 studies are published for all years in the range. Even in recent years, like 2016 (3) we see papers published. However, looking closely we find that these studies

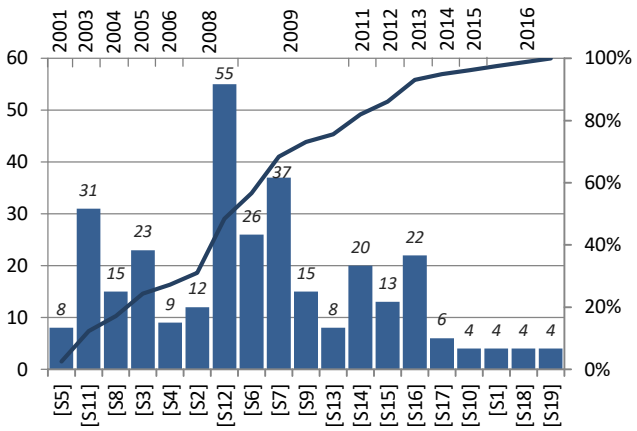

Figure 5. Number of best practices per publication.

mainly confirm and only partly complement the preexisting corpus of best practices. As Fig. 5 indicates, the majority (76%) of best practices was reported before 2010. To us this is not surprising, because studies published before 2010 include foundational work on DSM [S13] and DSLs [S15]. It is rather unsurprising for a meta-study on best practices, that the spectrum of research types is rather narrow. As best practices are derived from practical experiences the studies reporting these practices often as well are experience reports.

There is no single form how best practices are presented. However, semi-formal presentation is prevalent (see Fig. 2). By semi-formal presentation, we mean that studies structure content or parts therefrom by itemizing best practices. Semantic grouping of best practices to categories is done, but is not standard. Single practices are often given a short, recognizable name followed by descriptions and arguments. 3 studies provide a more formal description of best practices by means of patterns. Patterns typically are described by name, a context, problem, solution and relation to other patterns. Anecdotal descriptions of best practices are mostly provided by studies whose main contribution is the application of DSM in a certain context and the reported best practices are more of a side note to this. As the style of presentation already suggests, the way how advice is given varies between studies. Apart from worst practices, Fig. 3 shows an even distribution of how practices are reported. Moreover, Fig. 3 and Fig. 2 show that studies reporting on best practices, worst practices, lessons learned, or success factors are less formalized, and that patterns and guidelines choose a more formalized and structured presentation of practices.

B. Research Question 2: The Sources of Best Practices

To assess best practices it is insightful from which context authors derive best practices. Table IV lists four different sources, i.e. *industrial projects*, *literature studies*, *author's experience*, and *DSL tool development*. Five papers condense knowledge from base literature and publications in the scientific field. Studies referenced by these primary studies may again be derived from various sources, including reports on project experience, but do not contribute new project experiences. Moreover, the source of four studies can be classified as author's experience. Obviously, all authors of selected studies have experience in the relevant scientific field. However, we only classified a source as *author's experience* if this is explicitly stated in the paper or no other source is given in the publication. Expectantly, the majority (10 out of 19) of best practices come from industrial projects. An industrial setting, the need for constant improvement, and the adoption of a new development methodology are a good frame to deliver practices. To us it is interesting to see from how many projects experiences are distilled. In general, best practices condensed from many projects are more applicable to new projects than those derived from single projects only.

With exceptions, studies list between one and three projects. In [S13], Kelly and Tolvanen report 30 application domains in which they have applied DSM over a period of 15 years. This work clearly is foundational for DSM in general and provides a major starting point for practical guidance in DSM. Moreover, Kelly and Pohjonen [S10] report worst practices from 76 DSM cases.

The importance of tooling for both, the usage and the development of domain-specific modeling solutions, is reflected by recognizing *DSL tool development* as a source of best practices. Studies generally confirm the importance of tooling, and moreover, two studies, i.e. [S13] and [S12], partly derive their best practices from tool development for DSL and DSM. For instance, [S12] provides a profound body of patterns for the development of components of a language compiler (i.e. parser, transformations, internal representation, or generators).

C. Research Question 3: Specificity of Best Practices

To further categorize studies reporting best practices we analyzed contextual information for which best practices are reported. We chose language notation, users of a DSM solution, and the software development life-cycle phase as major cornerstones for this classification. A fundamental decision in the development of a DSM solution is whether to use a textual or graphical notation (see Table V). We assumed that studies, and therefore best practices, will be disjunctive on this binary classification. Surprisingly, it turned out that seven of our primary studies provide advice for both forms of notation. Similarly, we expected that best practices specifically turn towards business users (cf. Table VI). However, only four studies exclusively provide best practices from DSM solutions designed for business users only. All other 15 studies report best practices, that at least to some degree, refer to developers.

Table VII and Fig. 4 show which phases of the software development life-cycle are addressed by the selected studies. Noteworthy, all phases are addressed by experience papers which can be interpreted as strong indicator that in all phases challenges arise which are very specific for DSM. Data clearly show that most studies provide best practices for design and implementation. Fig. 4 reveals that this majority not only stems from experience reports but also from philosophical papers and evaluation research. Analysis and requirement gathering is the third-most covered phase. Mernik et al. [S4], classified as philosophical paper, provide in-depth advice on domain analysis methodologies, e.g. Feature Oriented Domain Analysis (FODA), to be used in DSL development.

Four papers contain best practices for testing DSM solutions, varying from general advice like *do not forget testing* to specific guidance for testing code generators. The paper of Ratiu and Völter [S19] is worth mentioning because they go beyond general advice and provide best practices for test automation of individual DSM components. Furthermore, they also emphasize the differences between testing compilers for general-purpose languages and code generators for DSLs. The deployment phase is covered very weakly, not only by numbers of publications but also by their strength of guidance for this phase. The two papers listed, only briefly mention that deployment must be supported. Similarly, maintenance is only weakly covered by [S1] and generally by the two as base literature considered studies [S13] and [S15].

D. Research Question 4: Best Practices

We distilled 130 best practices and 59 patterns from a total of 309 best practices and patterns found across all studies. Tables VIII, and IX list and group best practices by development phases in descending order with respect to the number of occurrences of a best practice across different studies. As merging of best practices involved generalization of best practices, even a single study may report a best practice multiple times from different perspectives. In general, we assigned best practices to the following phases *Planning*, *Analysis*, *Design*, *Implementation*, *Test*, and *Maintenance*. This categorization is not congruent to the categorization of studies in Table VII but adapted for the analysis of best practices. Process and organization-related practices are mostly assigned to the planning phase of DSM development. The majority of practices are grouped within phases *Implementation* (59) and *Design* (51) followed by *Analysis* (7), *Planning* (6), *Maintenance* (6), and *Test* (4). Patterns, reported in studies [S4], [S6], [S12], [S2], [S15], are not included in Tables VIII, and IX but summarized in Table X.

To answer which DSM-topics are addressed by single best practices we identified the parts of a DSM-solution which are targeted by best practices. Recognized parts are: domain model (DM), language design and concepts (LA), language notation (LN), generators (GN), DSL-tooling (DT), meta-model tooling (MT), and the entire DSM-solution (All). Best practices which are assigned to phase *Implementation* mainly target language implementation (13), tooling (12) and generative aspects (22). With respect to tooling importance of adequate tool support is stressed for both, the development of a DSM solution and its usage. Unsurprisingly, complexity of generator implementation is addressed by best practices. In particular, advices on modularization and purpose orientation of generators and quality and efficiency of generated code are provided. Best practices attributed to the *Design* phase address design of language concepts (37) and language notation (17). In testing best practices recognize language implementation and implementation of generators as targets. Maintenance issues address language design and user training. In *Planning* best practice provide advice for organizational and social-technical aspect of DSM-development which concern the entire DSM-solution.

Table VIII
BEST PRACTICES EXTRACTED FROM STUDIES

	Best Practice	Studies	Part
Plan	Domain engineering team	[S3] [S8] [S9] [S1] [S17]	All
	Iterative development	[S8] [S11] [S1] [S17]	All
	DSM requires organization	[S13]	All
	Treat DSM as product development	[S8]	All
	Aquire language development expertise	[S3]	All
	Fit technology transfer to business model	[S9]	All
Analysis	Define domain scope	[S3] [S5] [S13]	DM
	Identify DSL usage	[S7] [S9]	DM
	Use end-user personas	[S17]	All
	Perform expert review	[S17]	All
	Do not overestimate domain experts	[S9]	All
	Understand present solution	[S9]	DM
Design	Balance genericity and specialization	[S7] [S8] [S9] [S10] [S1] [S16]	LA
	Reuse language definitions	[S4] [S7] [S8] [S10]	LA
	Adopt existing domain notations	[S4] [S7] [S8] [S9] [S13] [S16]	LN
	Support model reuse on language level	[S7] [S10] [S13]	LA
	Design must have purpose	[S3] [S7] [S9] [S18]	LA
	Design for language evolution	[S8] [S10] [S11] [S1]	LA
	Carefully choose form of notation	[S7] [S8] [S10] [S13]	LN
	Avoid redundancy	[S5] [S7] [S18]	LA
	Viewpoint orientation	[S3] [S8] [S13]	LA
	Provide Integrability	[S5] [S13]	LA
	Make elements distinquishable	[S7] [S10] [S13]	LN
	Consistent style everywhere	[S7] [S13]	LN
	Limit nr. of language elements	[S7] [S11]	LA
	Support variability on language level	[S13]	LA
	Reuse type systems	[S4] [S7]	LA
	Keep it simple	[S5] [S7]	LA
	Effective process for DSML definition	[S3] [S4]	LA
	Care for quality	[S5] [S13]	LA
	Perform Care for usability evaluations	[S17]	All
	Create guidelines for language design	[S13]	LA
	Define language rules	[S13]	LA
	Derive concepts from physical sturcture	[S13]	LA
	Derive concepts from Look&Feel	[S13]	LA
	First things first - the language	[S13]	LA
	Derive concepts from domain	[S13]	LA
	Derive concepts from expected output	[S13]	LA
	Reuse existing computational model	[S13]	LA
	Balance compactness and understd.	[S7]	LA
	Interface concept	[S7]	LA
	Precise high-level abstraction	[S1]	LA
	Detect recurring patterns in design	[S1]	LA
	Compose existing languages	[S7]	LA
	Provide sufficient level of detail	[S16]	LA
	Multiple levels of abstractions	[S16]	LA
	Clear language to target mapping	[S16]	LA
	Turn API into DSL	[S4]	LA
	Strive for 80% solution	[S9]	LA
	Care for longevity	[S5]	LA
	Care for scalabiltiy	[S5]	LA
	Care for usability	[S5]	LA
	Beware of misplaced emphasis	[S10]	LA
	A library is not the language	[S10]	LA
	Filter details from notational elements	[S13]	LN
	Use colors	[S13]	LN
	Use transparency containments	[S13]	LN
	Derive notation from corporate identity	[S13]	LN
	Ask graphic designers for help	[S13]	LN
	Use descriptive notations	[S7]	LN
	Usage of syntactic sugar	[S7]	LN
	Transform visual to text	[S4]	LN
	Use mixtuer of language notations	[S9]	LN

Table IX
BEST PRACTICES EXTRACTED FROM STUDIES

	Best Practice	Studies	Part
Implementation	Importance of meta-tooling	[S3] [S4] [S8] [S10] [S13] [S14] [S16]	MT
	Importance of DSL-tooling	[S3] [S5] [S8] [S13] [S14] [S1]	DT
	Use generators of varing granularity	[S13]	GN
	Incremental development	[S8] [S10]	All
	Partition your meta-model	[S3] [S8] [S14]	LA
	Don't modify generated code	[S8] [S1]	GN
	Simplify generators	[S8] [S13]	GN
	Use source-source transformation	[S4] [S6]	GN
	Restrict host language by specialization	[S4] [S6]	LA
	Provide meta-metamodel	[S13] [S16]	MT
	Use constraint language in Meta-modeling	[S16] [S18]	MT
	Create proof of concepts to show benefits	[S13]	All
	Follow inhouse standards	[S13]	All
	Document your code	[S8]	All
	Plan DSL usage reviews	[S8]	All
	Identify DSL usage process	[S10]	All
	Move duplicated code to domain framework	[S13]	DF
	Provide good API for generated code	[S13]	DF
	Domain specific frameworks	[S8]	DF
	Hide complexity in domain framework	[S9] [S13]	DF
	Provide user modes	[S13]	DT
	Identify usage convention	[S7]	DT
	Teamwork support	[S8]	DT
	Check constraints first	[S8]	DT
	Avoid debugging at source-code-level	[S10]	DT
	Turn DSM models into documenation	[S13]	GN
	Integrate handwritten code on file level	[S13]	GN
	Use autobuild for DSM models	[S13]	GN
	Parallel generators per prog. langugage	[S13]	GN
	One version for all	[S13]	GN
	Do not build generator to soon	[S13]	GN
	Generate all documents	[S14]	GN
	Stabilize design quickly	[S1]	GN
	Modularize transformations	[S1]	GN
	Consistent input to transformations	[S1]	GN
	Interpretation vs. code generation	[S8]	GN
	Control manually written code	[S8]	GN
	Care about generated code	[S8]	GN
	Make code true to the model	[S8]	GN
	Viewpoint aware processing	[S8]	GN
	Care about templates	[S8]	GN
	Effective Generators	[S3]	GN
	Avoid semiautomatic transformation	[S10]	GN
	Reuse mechanisms in metamodel	[S13]	LA
	Model sharing and splitting mechanisms	[S13]	LA
	Version your models	[S13]	LA
	Use DSL to enforce standards	[S14]	LA
	Syntax alignment of concepts	[S7]	LA
	Allow for adaptation	[S8]	LA
	Create annotation models	[S8]	LA
	Test your DSL	[S8]	LA
	Base implementation on usage	[S4]	LA
	Avoid enforcing validation rules	[S10]	LA
	Separate language model from validation	[S11]	LA
	Syntax agnostic layouts	[S7]	LN
	Instance representation for meta-models	[S16]	MT
	Common graphical notation for meta-tooling	[S16]	MT
Maint.	Provide training	[S10] [S13] [S14]	All
	Understand and deal with your weaknesses	[S9]	All
	Users do not overlook your mistakes	[S9]	All
	Use tooling for model evolution	[S13]	LA
	Plan for evolution	[S13]	LA
	Expect redesign and plan	[S1]	LA
Test	Define cost-benefit of generator testing	[S19]	GN
	Use fuzzy testing	[S19]	GN
	Measure test coverage	[S19]	GN
	Create DSLs for testing DSLs	[S19]	LA

All patterns extracted from studies map to phases *Design* and *Implementation* and target similar DSM-topics as best practices do. Obviously, patterns provide a structured and precise description of when and how to apply the pattern to DSM development. Patterns mainly target technical aspects in the development of textual languages, i.e. parser development and input processing, the creation and processing of meta-models, design and implementation of type systems, and generator development. Patterns extracted from selected studies tend to address topics on a rather fine-grained level (E.g. a symbol table pattern for tracking language elements). Therefore, and for the sake of brevity we aggregate patterns

Table X
PATTERNS EXTRACTED FROM STUDIES

Pattern Category	Studies	Nr.
Parser development	[S12] [S15]	17
Meta-modeling	[S2] [S6] [S12] [S15]	14
Processing meta-models	[S4] [S6] [S12] [S15]	11
Interpreter pattern	[S12]	6
Language implementation	[S4] [S6] [S15]	4
Typing	[S12]	4
Generator pattern	[S12]	3

to categories. Categories are presented in Table X.

A major point of interest to us was how reported practices match and complement each other. With respect to studies reporting the same best practices, we find most consensuses in the categories *Planning*, *Analysis*, and *Design*. Low consensus can be found in category *Implementation*. For *Maintenance* and *Test* best practices are collected from too few studies to make an evaluation. In the category *Design* we identified 17 from a total of 51 best practices that are supported by more that one author. Most consensus was found on best practices which reflect on domain concepts in language design, redundancy, simplicity, distinguishability, and choosing between generic and specific notations. Also notable, a third of best practices (17 from 51) of this category are identified by a single study [S7]. Best practices for category *Implementation* are rather complementing than matching. Only, the importance of tooling for both, development of a DSM-solution (i.e. meta-tooling) and tool support for end-users, is identified multiple times. This is not surprising, as we find a common consensus that DSM solutions are best developed by means of language workbenches, not only in papers providing best practices but also by general DSM literature [1]. The majority in this category is identified by [S8]. Before the in-depth study of papers, we expected to find many contradictions between best practices. A reason for that assumption was that best practices were collected from different real projects with possibly conflicting requirements. However, only one contradictions could be identified. While Mernik et. al. in [S4] report on the possibility to turn an API into a DSL, Kelly and Pohjonen in [S10] advice developers to restrain from doing so. Their worst practice *the library is the language* states that deriving a language from frameworks leads to low abstraction level and opposes fundamental ideas of DSM.

V. THREATS TO VALIDITY

This systematic mapping study has similar threats than other systematic mapping studies. Selection of digital libraries and creation of search queries are seen as threat to validity. We tried to address these issues by selecting digital libraries most important in the field of computer science and supplemented results by using web-search engines and meta-search engines. To mitigate the risk of missing out papers due to incomplete search queries we iteratively refined search terms, starting with an initial set derived from base literature. However, we cannot rule out that we missed studies that report best practices, due to our search strategy to search document titles only. Exclusion of relevant studies during screening is another threat to validity. We tried to mitigate this by clearly defined criteria and a multistage screening process. However, for papers which report practices in an anecdotal way we cannot rule this out. An obvious threat to validity concerns the extraction of best practices from single studies. Extraction was unequivocal for studies which list best practices in a formal or semi-formal manner. However, for studies which present best practices in a narrative or anecdotal way extraction included analysis and interpretation by the authors of this paper. Moreover, merging practices from different studies required comparison, interpretation and abstraction by the authors. We mitigate this threat by providing traceability from best practices as found to best practices as presented in the final compilation.

VI. CONCLUSION

In this paper we presented the results of a systematic mapping study on best practices for domain-specific modeling. We selected 19 papers which report best practices, guidelines, and patterns addressing specific challenges in the development of DSM solutions. We answered three research questions (RQ-1 to RQ-3) which map and classify best practice reports in the field of domain-specific modeling. Furthermore, we answer (RQ-4) which best practices are reported by providing a compact but comprehensive compilation of 189 unique best practices. We found that literature provides best practices which mainly complement each other, sometimes match other practices, and hardly contradict each other. Moreover, it is interesting to see that best practices are provided for all phases of the software development cycle, whereby most of which focus on design and implementation. More work would clearly be desirable for maintenance and test of DSM-solutions. The compact compilation of best practices provided in this paper may act as a starting point to identify literature which facilitate the industrial adoption of DSM practices. Obviously, a more detailed and comprehensible presentation of best practices would be desirable. To create this comprehensive compilation and make it publicly available clearly is the intention of the authors of this paper. The assessment of usefulness, importance, and finally impact of reported best practices was out of scope of this study. However, future work will investigate the perception of reported best practices in industry. By this we hope to provide even stronger support for industrial adoption of DSM and identify topics not yet covered by research.

ACKNOWLEDGMENT

The research reported in this paper has been supported by the Austrian Ministry for Transport, Innovation and

Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

Primary Studies

[S1] S. Livengood, "Experiences in domain-specific modeling for interface specification and development," in *Proceedings of the 2Nd International Master Class on Model-Driven Engineering: Modeling Wizards*, ser. MW '12. New York, NY, USA: ACM, 2012, pp. 4:1–4:2.

[S2] J. Iber, A. Höller, T. Rauter, and C. Kreiner, "Patterns for designing configurability into domain-specific language elements," in *Proceedings of the 21st European Conference on Pattern Languages of Programs*, ser. EuroPlop '16. New York, NY, USA: ACM, 2016, pp. 1:1–1:14.

[S3] A. Kahlaoui, A. Abran, and E. Lefebvre, "Dsml success factors and their assessment criteria," *Metrics News*, vol. 13, no. 1, pp. 43–51, 2008.

[S4] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, dec 2005.

[S5] D. S. Kolovos, R. F. Paige, T. Kelly, and F. A. Polack, "Requirements for domain-specific languages," in *Proc. of ECOOP Workshop on Domain-Specific Program Development (DSPD)*, vol. 2006, 2006.

[S6] D. Spinellis, "Notable design patterns for domain-specific languages," *J. Syst. Softw.*, vol. 56, no. 1, pp. 91–99, Feb. 2001.

[S7] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schneider, and S. Völkel, "Design guidelines for domain specific languages," in *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09)*, M. Rossi, J. Sprinkle, J. Gray, and J.-P. Tolvanen, Eds., 2009, pp. 7–13.

[S8] M. Völter, "Best Practices for DSLs and Model-Driven Development," *Journal of Object Technology*, vol. 8, no. 6, pp. 79–102, Sep. 2009.

[S9] D. Wile, "Lessons learned from real dsl experiments," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. IEEE, 2003, pp. 10–pp.

[S10] S. Kelly and R. Pohjonen, "Worst practices for domain-specific modeling," *IEEE software*, vol. 26, no. 4, 2009.

[S11] M. Vierhauser, R. Rabiser, P. Grünbacher, and A. Egyed, "Developing a dsl-based approach for event-based monitoring of systems of systems: Experiences and lessons learned (e)," in *30th IEEE/ACM International Conference on Automated Software Engineering, Lincoln, USA*, 2015, pp. 715–725.

[S12] T. Parr, *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*, 1st ed. Pragmatic Bookshelf, 2009.

[S13] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley, 2008.

[S14] M. Vokáč and J. M. Glattetre, "Using a domain-specific language and custom tools to model a multi-tier service-oriented application — experiences and challenges," in *Model Driven Engineering Languages and Systems*, L. Briand and C. Williams, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 492–506.

[S15] M. Fowler, *Domain Specific Languages*, 1st ed. Addison-Wesley Professional, 2010.

[S16] U. Frank, *Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 133–157.

[S17] M. Moser, M. Pfeiffer, and J. Pichler, "Domain-specific modeling in industrial automation: Challenges and experiences," in *Proceedings of the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation*, ser. MoSEMInA 2014. New York, NY, USA: ACM, 2014, pp. 42–51.

[S18] P. Salehi, A. Hamou-Lhadj, M. Toeroe, and F. Khendek, "A uml-based domain specific modeling language for service availability management," *Comput. Stand. Interfaces*, vol. 44, no. C, pp. 63–83, Feb. 2016.

[S19] D. Ratiu and M. Völter, "Automated testing of dsl implementations: Experiences from building mbeddr," in *Proceedings of the 11th International Workshop on Automation of Software Test*, ser. AST '16. New York, NY, USA: ACM, 2016, pp. 15–21.

References

[1] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley, 2008.

[2] A. van Deursen and P. Klint, "Little languages: Little maintenance," *Journal of Software Maintenance*, vol. 10, no. 2, pp. 75–92, Mar. 1998.

[3] D. A. Ladd and J. C. Ramming, "Two application languages in software production," in *Proceedings of the USENIX Symposium on Very High Level Languages*. USENIX, 1994, pp. 169–187.

[4] E. Evans, *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[5] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers principles, techniques, and tools*. Reading, MA: Addison-Wesley, 1986.

[6] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE'08. Swindon, UK: BCS Learning & Development Ltd., 2008, pp. 68–77.

[7] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - a systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2008.09.009

[8] D. Budgen, M. Turner, P. Brereton, and B. A. Kitchenham, "Using mapping studies in software engineering," 2008.

[9] M. Fowler, *Domain Specific Languages*, 1st ed. Addison-Wesley Professional, 2010.

[10] O. Olajubu, "A textual domain specific language for requirement modelling," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 1060–1062.

[11] R. Wieringa, N. A. M. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requirements Engineering*, vol. 11, pp. 102–107, 03 2006.

[12] W. W. Royce, "Managing the development of large software systems: Concepts and techniques," in *Proceedings of the 9th International Conference on Software Engineering*, ser. ICSE '87. Los Alamitos, CA, USA: IEEE Computer Society Press, 1987, pp. 328–338.