Survey of Established Practices in the Life Cycle of Domain-Specific Languages

Holger Stadel Borum
hstb@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

Christoph Seidl
chse@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

ABSTRACT

Domain-specific languages (DSLs) have demonstrated their usefulness within many domains such as finance, robotics, and telecommunication. This success has been exemplified by the publication of a wide range of articles regarding specific DSLs and their merits in terms of improved software quality, programmer efficiency, security, etc. However, there is little public information on what happens to these DSLs after they are developed and published. The lack of information makes it difficult for a DSL practitioner or tool creator to identify trends, current practices, and issues within the field. In this paper, we seek to establish the current state of a DSL's life cycle by analysing 30 questionnaire answers from DSL authors on the design and development, launch, evolution, and end of life of their DSL. On this empirical foundation, we make six recommendations to DSL practitioners, scholars, and tool creators on the subjects of user involvement in the design process, DSL evolution, and the end of life of DSLs.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**;
• **General and reference → Surveys and overviews**.

KEYWORDS

Domain-specific languages, Survey

ACM Reference Format:
Holger Stadel Borum and Christoph Seidl. 2022. Survey of Established Practices in the Life Cycle of Domain-Specific Languages. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3550355.3552413

1 INTRODUCTION

Domain-specific languages (DSLs) have been established as a multi-faceted tool in software engineering that can be used to improve usability [17], performance [27], security [25], and code reuse [19]. From an academic standpoint, little is known about most DSLs' life cycle beyond initial publications that demonstrate their merits, whether they be innovations within the application domain [20], investigations and innovations of DSL technologies [40], or more processual design considerations [12]. Reports on the broader life cycle of DSLs are scarce, if not altogether lacking. While some experts may have enough experience creating DSLs to know the current state of practices, it should be available to any DSL practitioner and tool-creator. Kosar et al. find in their systematic mapping study that most primary DSL studies focus on domain analysis, design and implementation, with only 10 out of 390 investigating maintenance and validation [24]. We find that the articles closest to investigating the DSL life cycle typically fall into two categories. First, a retrospective analysis summarises lessons learned from years of working with DSLs [23, 41]. Second, DSL evolution may be investigated through an explicit process discussion of evolution reasons and results [18, 35, 37, 39]. While both types of investigations are valuable and should be encouraged, their scarcity, combined with their diverse and non-standardised reporting, makes it difficult to use them for establishing the current state of DSLs' life cycles.

In this paper, we set out to alleviate the lack of information on DSLs after their publication by empirically investigating the established practices in the life cycles of DSLs. Our purpose is to survey current established practices in DSLs' life cycles and, from this foundation, make recommendations for future research. We do so through a questionnaire sent to selected DSL authors on their DSLs' life cycle phases (design and development, launch, evolution, and end of life). The questionnaire has an effective response rate of 43% from DSL authors whose DSL appeared in eight different DSL corpora (Section 2). The questionnaire is especially focused on topics difficult to examine outside of self-report since other methods more easily investigate technical choices such as tooling, see [21]. This focus translates to questions on user perspectives in different life phases. In addition, the survey investigates other aspects such as development setting, causes of DSL evolution, and reflections on the impact of different initiatives (Section 3). Based on our analysis, we make six recommendations for the DSL field (Section 4). We discuss threats to the validity of our findings (Section 5) and the utility of our approach compared to similar research (Section 6). To conclude, we summarise our findings (Section 7).

The contributions of this paper are:

- The presentation of empirical data regarding the design and development, launch, evolution, and end of life of DSLs.
- An analysis of data establishing current practices in the life cycles of DSLs.
- Empirically based recommendations for DSL practitioners.

MODELS '22, October 23–28, 2022, Montreal, Canada

ACM ISBN 978-1-4503-9466-6/22/10…$15.00
https://doi.org/10.1145/3550355.3552413

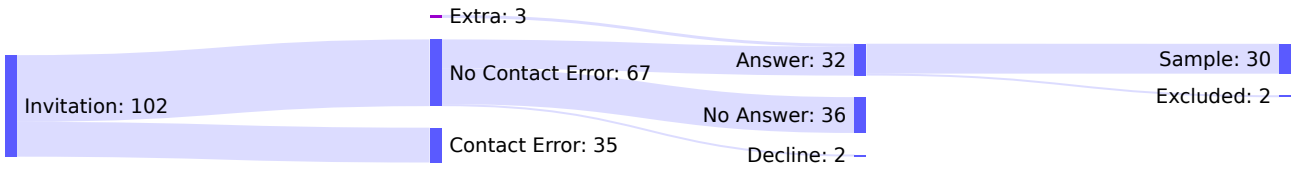

Figure 1: A depiction of our process flowing from sending invitations to the left to obtaining the sample to the right.

2 METHOD

The purpose of our survey was to establish current practices in DSL's life cycle by sending a questionnaire [13] to DSL authors. The fundamental method for our investigation was to first identify a number of relevant DSLs, then send the authors of these DSLs an invitation to the questionnaire regarding their specific DSL, and, finally, analyse the responses. We sought to obtain a representative sample of DSLs of interest to academia and industry by surveying DSLs from curated DSL collections. We sent the first round of invitations during the Summer of 2021, and we sent a second round of re-invitations during the Winter of 2021 to authors we had been unable to contact. We postpone discussions of methodological threats to validity to Section 5.

We asked all invitees to answer the questionnaire regarding a specific DSL project through email or a similar contact method. Figure 1 shows how the original 102 invitations resulted in a sample of 30. We received an error message from 35 of our invitations, meaning that 67 DSL authors received an invitation at most. We allowed participants to make submissions about other DSL projects they found relevant. We received and included three such submissions, which we label as *extra*. We obtain an effective response rate of at least $29/67 = 43\%$ when not counting *extra* answers.

2.1 Invitations

We used existing curated collections of DSLs as our pool of invitation candidates. We chose the following collections to ensure that the investigated DSLs were of interest to academia and industry:

- The paper, *Domain-Specific Languages: An Annotated Bibliography* [39]
- The workshop series proceedings of *Real World Domain Specific Languages* [3, 4, 6, 7]
- The conference proceedings of *Domain-Specific Languages '99* [1]
- The workshop proceedings of *Functional Programming Concepts in Domain-Specific Languages* [2]
- The workshop proceedings of *Domain-Specific Languages Design and Implementation* [38]
- The collection of financial DSLs, *dslfin.org* [10]
- The collection of DSLs, *Wikipedia categories* [5]
- The collection of robotic DSLs, *Robotics DSL Zoo* [9, 30]

We had two exclusion criteria for filtering out artefacts appearing in these collections. We removed an artefact if it a) was not a DSL or if b) the authors did not intend for their DSL to be used. Criterion *a* was primarily important since these collections contain many methods and tools not relevant to the survey. When making this decision, we considered 1) if it was stated that the artefact was a DSL, 2) if the artefact had textual or visual language constructs tailored to a specified domain, and 3) if the application domain of the artefact was narrow. We primarily removed non-language artefacts, but we removed a couple of artefacts since we analysed them to have too many general-purpose characteristics to include. One example of such a language is the programming language Q# [36], designed for expressing quantum algorithms seamlessly alongside classical computations. While the domain of Q# is quantum algorithms, we find that its domain application domain of quantum computations and classical computations is an extension of many general-purpose languages. For criterion *b*, we were lenient in deciding whether a DSL was not intended to be used by only excluding DSLs where the authors explicitly stated they were not. This leniency is also apparent from the received answers, where two participants explicitly stated that the purpose of the DSL was not intended for use. We excluded these submissions from the final sample.

2.2 Questions and Data Interpretation

Besides factual questions, the questionnaire consisted of a) multiple-choice questions allowing multiple answers and free-text answers and b) free-text questions. Only a few factual questions were required to be answered by participants. All of these questions were easily answerable by all participants to not create a barrier to participation.

Free-text answers were analysed and grouped into representative categories using open coding. We either created a new category or used a predefined choice when appearing in multiple-choice, multiple-answers questions. To be transparent about this process, we emphasise new categories in figures, and for each category, we visualise the fraction we interpreted to belong there. We used only new categories when interpreting answers to free-text questions since there were no predefined choices for these questions. We assigned a single free-text answer to several categories when we found it to describe a multitude of subjects. In the presentation, we show representative cases for each category. These answers have been anonymised and edited for presentation while preserving their original meaning. We excluded a couple of free-text answers from interpretations since they contained apparent mistakes. When we encountered such answers, we ensured that the remaining answers of the submission were coherent.

3 RESULTS

The survey questionnaire focused on five topics: 1) DSL characteristics, 2) evaluation and user involvement in design, 3) launch, 4)

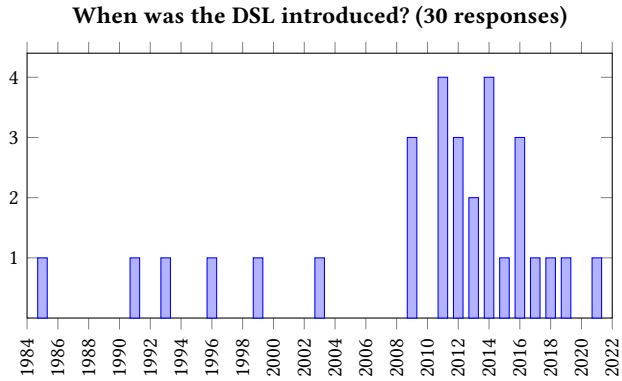

Figure 2: Data plot of when the surveyed DSLs where introduced. The mean age of the DSLs is 11 (introduced in 2010).

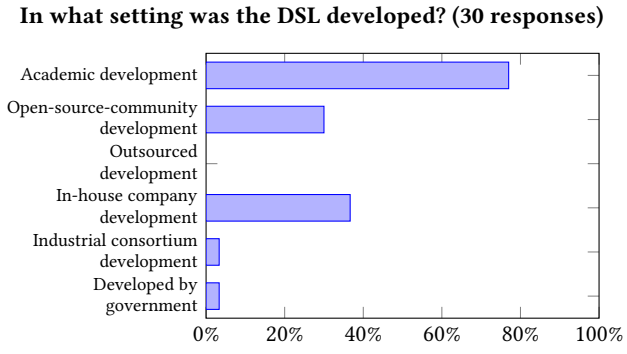

Figure 3: Data plot of development setting of surveyed DSLs.

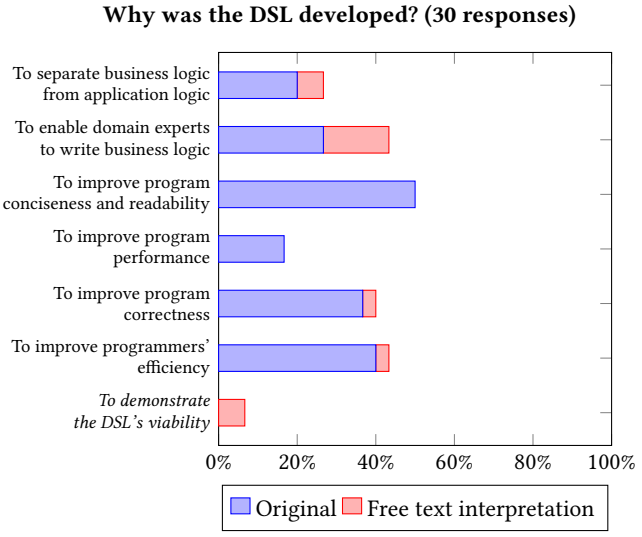

Figure 4: Data plot of reasons for developing the DSL.

evolution, and 5) end of life. We treat each topic in its own section by first giving an executive summary, then presenting the results, and finally discussing our findings. We cross-examine data from different topics whenever relevant. We report answers to multiple-choice questions in percentages and answers to free-text questions in the number of respondents. Findings and claims are numbered using circled numbers ⓧ which permit later referencing.

3.1 DSL Characteristics

We present the characteristics of the 30 DSLs in our sample, i.e., the age of the DSLs, their development setting, and the reason for creating them[1]. The characterisation of the sample forms a basis for the kind of DSLs our subsequent findings are generalisable to. The surveyed DSLs are predominantly developed in an academic setting and are on average 11 years old, with the majority being younger.

Results. The DSLs in the sample are between 0 and 36 years old (introduced between 1985 and 2021), with an average age of 11 years. While this is a broad range, 80% of the DSLs are younger than 12 years (Figure 2). Of the surveyed DSLs, 77% were developed within an academic setting (Figure 3). Still, of these DSLs developed in an academic setting, 52% were also partly developed within some other setting. Of the surveyed DSLs, 30% were developed in an open-source community, 38% were developed in an industrial setting[2], and 19% were developed in both an academic and industrial setting. The sample represents all of our suggested reasons for developing a DSL (Figure 4). While 50% of DSLs sought "to improve program conciseness and readability", only 3% had this as their only reason. Comparatively, 55% of DSLs were created for more technical reasons such as "to separate business logic from application logic" or "to improve program performance".

Discussion. We confirmed our anticipation of having a majority of academically developed DSLs since we primarily invited DSLs from academic collections (see Section 2). We find that while it is often mentioned that DSLs allow for a higher-level description of a domain, this is rarely the only reason for creating a DSL. Authors often seek other effects allowed by the higher-level description, such as improving program correctness or programming efficiency. In all, ① we find that the sampled DSLs primarily represent academically developed DSLs with some industrial uses that were developed for many different reasons and have an age between 5 and 12 years.

3.2 Evaluation and User Involvement

Expert evaluation is the most used evaluative method during DSL design, but different user-centred evaluation methods are also commonly used. We find no established practice for the level of user involvement during the DSL design, and we do not find any correlation between the degree of user involvement and a DSL's introduction year or the level of users' programming experience.

Results. ② Expert evaluation is the predominant method for evaluating a DSL during design and development, and it is used by 57% of DSLs (Figure 5). Of these, 41% use it as their only evaluative method.

[1]We are also interested in the type of users presented in Section 3.2.
[2]This category consists of in-house development, industrial consortium development, and governmental development.

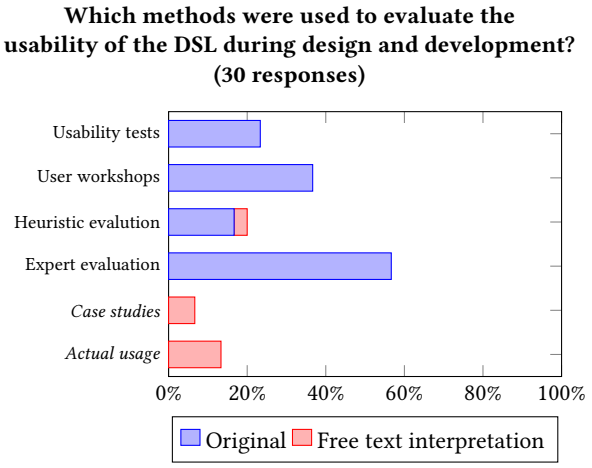

Figure 5: Data plot of broad categories of evaluative methods. We have shown free-text answers mentioning different kind of case-studies as their own category, although they could also be considered as a form of expert or heuristic evaluation depending on how they were conducted.

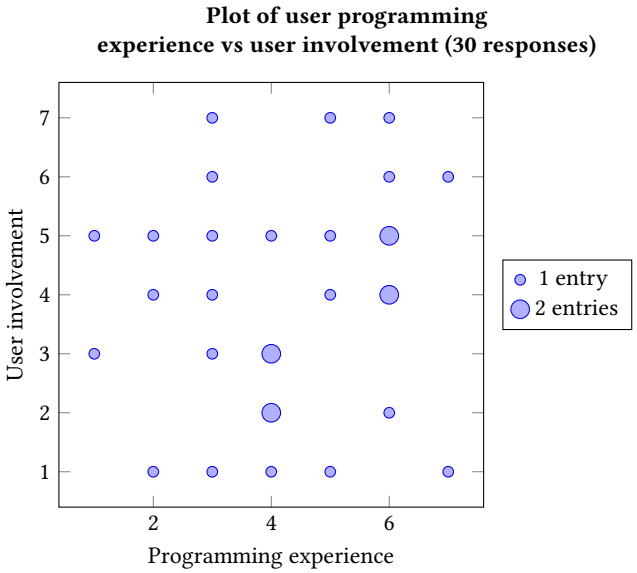

Figure 6: A data plot of users' programming experience vs. the level of user involvement in DSL design. The dimensions range from none (score 1) to expertise/involvment in all decisions (score 7).

Different human-centred evaluation methods[3] are used by 53% of DSLs. Finally, ③ heuristic evaluation is used only by 20% of DSLs, even though the method does not require any user involvement.

We find that DSLs are created for target users who have vastly different levels of programming experiences ranging from none (score 1) to expert (score 7) (Figure 6). While there are DSLs that target users with no programming experience, it is more common for DSLs to target users that have at least some programming experience. We also find that there is a wide range of involvement of users in the design process ranging from none (score 1) to involvement in every decision (score 7) (Figure 6). ④ Users are primarily involved in designing the DSL by assisting in requirement elicitation and by giving feedback during user tests, workshops, or from actual usage (Table 1). For requirement elicitation, users are involved in outlining the domain of the DSL, either by telling what kind of problems they usually face, by more actively designing scenarios, or by being involved in designing the DSL itself. The dominating methods for obtaining feedback on a DSL design are testing with users, conducting workshops, and actual language usage. Six respondents mentioned that these techniques are used iteratively through several increments of the DSL. Different kinds of language demonstrations are also used to obtain feedback from users through iterative interactions. It is common for DSLs to be designed by their users, either by having the designer being a user themself or by having a user within the design team.

Discussion. ⑤ We conjecture that the evaluative technique of expert evaluation has a significant impact on the usability of created DSLs due to its widespread usage. Also, we find that many DSL authors are interested in having users evaluate their DSL during design and development seen by the usage of human-centred methods.

For the programming experience of users, we expected most DSLs to be designed for users with some programming experience since we expected most DSLs to be developed for a domain where programming activities have become necessary. We find it more surprising that 17% of DSLs are created for users with little to no programming experience.

We find that there are large differences in how users are involved in the design project and the level of involvement ⑥. When the users of the DSL are part of the design team, we find it safe to presume that these in-team users are involved to some extent with most aspects of the language design and thereby impact the design significantly. We hypothesised that there could be a correlation between the level of user involvement in the design process and the programming experience of users. It is reasonable to involve users in the design of the DSL because they, as experts, are qualified to give informed feedback. On the contrary, it is also reasonable to involve users with little experience to ensure they understand the designed concepts and find them adequate. However, ⑦ the Kendall rank correlation coefficient of 0.09 between programming experience and user involvement shows no correlation with a p-value of 0.52. We made similar correlation tests between the year of DSL introduction and programming experience or user involvement. Both of these tests found no correlation with p-values of respectively 0.33 and 0.84.

[3]This category consists of usability tests and user workshops.

Table 1: How were users involved in the language design? (30 responses)

Requirement elicitation (7 answers)	• We asked the users to specify mathematically the problems they usually face. • Use case driven approach, elaborating on domain level specification of specific use cases.
Feedback (8 answers)	• Via conferences and mailing lists, where they could express their opinions. • Workshops for the initial and later incremental versions. • Feedback from a small in-house group. • Continuous user testing.
In team (4 answers)	• Prospective users were in the design team. • A small set of expert users were involved from the beginning in all aspects.
Designer is the user (3 answers)	• I am a user myself and was part of the creation and design of the language.
Actual/early usage (2 answers)	• Make them use the initially released versions; collect feedback to improve subsequent releases. • Users used the initially released version; they gave feedback for later releases.
Other (5 answers)	• Feature-Oriented Domain Analysis. • Through case studies.
No involvement (2 answers)	• The language is primarily machine-to-machine oriented.

3.3 Launching

DSL authors use a wide range of techniques to attract a user base when launching a DSL. Such techniques often affect the success of a DSL, and it is especially important to demonstrate the merits of using the language to attract users.

Results. The language ecosystem (development environment, training material, and language documentation) is used by 73% of DSLs to encourage usage. Smoothing the transitioning to using the DSL by either being backwards compatible or assisting in the transitioning is part of 30% of DSLs' launch strategy. Another approach used by 13% of DSLs is to require usage through company policy. Seven respondents opted to provide free-text answers focusing on promotion and social initiatives instead of the more technical approach suggested by the proposed answers.

Of 23 respondents, 9 answered that the initiatives for encouraging DSL usage affected their DSL's success (Table 2). Two topics reoccur in several responses. First, the initiatives are decisive for the success of the DSLs. Second, the assistance in transitioning from old solutions sometimes mitigates a steep learning curve. Again, seven respondents replied that the success of the DSL is primarily affected by its merits and the demonstration of these.

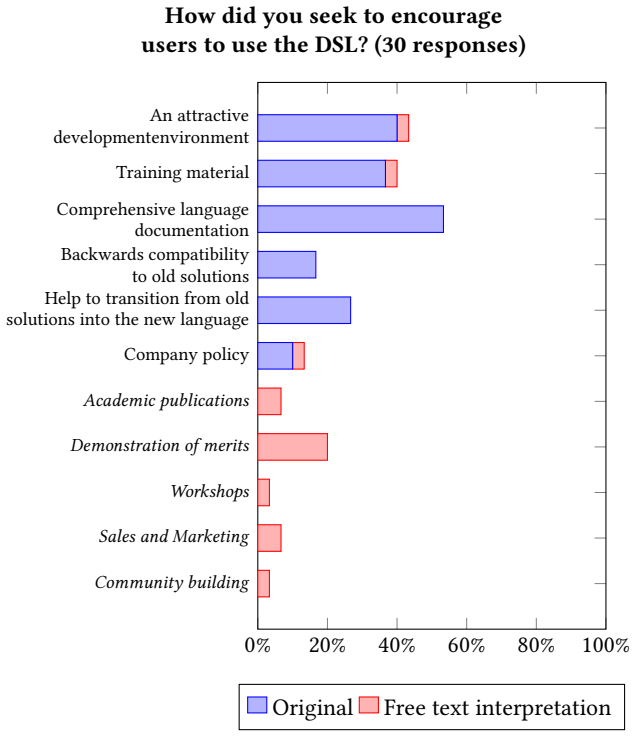

Figure 7: Data plot of techniques for encouraging DSL usage.

Discussion. While our proposed reasons for user encouragement primarily focus on implementation efforts, free-text answers focus more on demonstrating the merits of the DSL, social initiatives, and promotion. We would expect more participants to choose even more of such options if we had originally proposed them. We find that respondents emphasise that the merits of a DSL should present an improvement in the quality of life of its users compared to its alternatives. One respondent mentions that, for them, the improvement is so significant that the alternative of not using the DSL is unattractive. We interpret these answers relating to social initiatives and promotion to indicate that they see these aspects as necessary. Therefore we find that (8) it is essential to demonstrate the primary parts of the DSL to users, namely its merits. This demonstration can occur through different channels such as academic publications, sales and marketing, or community building.

3.4 Evolution

Almost all DSLs evolve after their launch, which is experienced by users as, among others, new language constructs and improvements in language implementation. How the evolution is performed is important for the success of a DSL, whether it is improving the existing language implementation, finding new application domains, or improving the development environment.

Results. (9) Evolution is a pervasive phenomenon for DSLs, with 86% of respondents of the entire sample reporting at least one cause of evolution. Almost all suggested evolution factors are equally common (Figure 8). Of respondents, 60% are affected by factors

Table 2: Do you think these efforts affected the success of the DSL? Why? (23 responses)

Affirmative (9 answers)	• Yes, but some users prefer to stick to the GUI. • Yes. I don't think people would have adopted it if we hadn't actively promoted and sold it. • Yes, the transition from earlier languages was effective.
Language merits important (7 answers)	• It succeeded really as a step-change in what could be done in terms of quality: there are many other DSLs in this general domain, but today only a few are meaningfully used. These present such opposite extremes that there is no real decision point in a single project as to which to use.
Other (7 answers)	• Users had by policy to use the DSL. • I would not characterise the DSL as very successful, but it does allow the users to quickly extend the existing system with non-standard products.

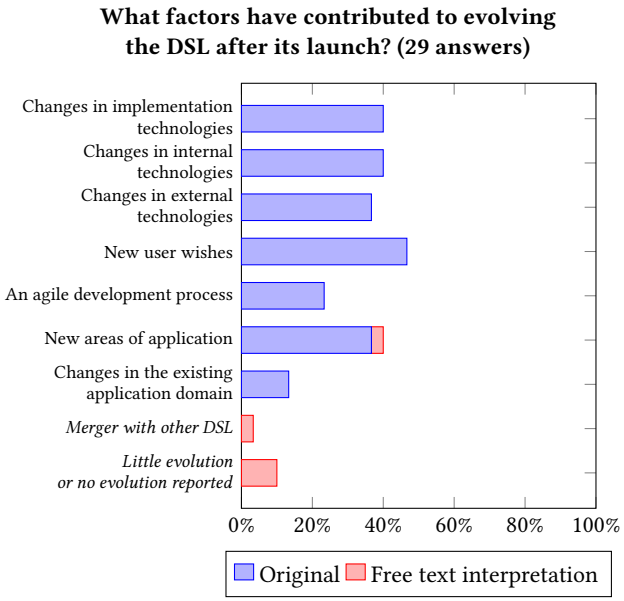

Figure 8: Data plot of evolution causes.

they are somewhat in control over, such as accommodating user wishes, having an agile development process, seeking new areas of application, and making changes to technologies internal to the maintainer. Still, 63% are affected by factors outside of their control, such as changes to the application domain, implementation technologies, and external technologies.

From the users' perspective, the most common form of DSL evolution is the addition of new language constructs or syntactic sugar (Figure 9). It is comparatively rare that major changes are made to the existing syntax, which happens to 23% of DSLs. (10) 17% of DSL creators have found it necessary to introduce breaking updates. A

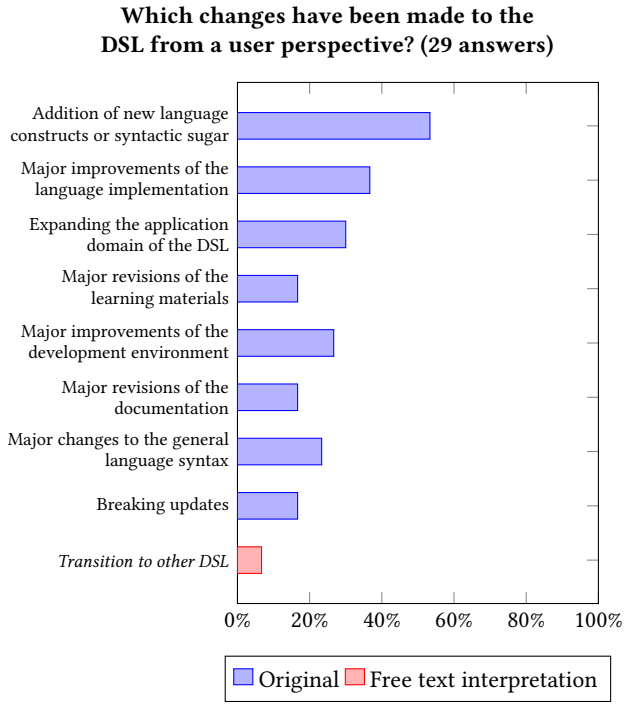

Figure 9: Data plot of DSL changes from the user perspective. Judging whether is major was left to the discretion of participants.

major evolution of the ecosystem of DSLs also occurs but is rarer.

(11) Of 21 respondents, 13 answered that evolution of a DSL is important, if not vital, to the success of DSLs (Table 3). As we have already shown, the evolution may take many forms, such as improving the quality of the DSL, accommodating user wishes, and making it more accessible and applicable. These improvements are important since they may introduce benefits that did not originally exist and help keep the DSL relevant. One respondent who mentions that evolution did not significantly affect their DSL's success even states that "I suspect those changes did not affect the language's success, except that if it had never made any progress, then it would have faded away."

Discussion. We identify that a DSL's constituents have different evolutionary characteristics. The questionnaire indicates that often the DSL (language constructs, implementation, and domain) itself is susceptible to evolution caused by new user wishes and domain evolution. Comparatively, the ecosystem of the DSL is less susceptible to major evolution. We hypothesise that this difference is due to the developers having more control over the ecosystem of the DSL than the usage of the DSL.

We expected that most DSL authors would rarely introduce breaking updates to their language. Surprisingly, we found that breaking updates are not that uncommon. We hypothesise that

Table 3: Do you think these efforts affected the success of the DSL? Why? (21 answers)

Vital (1 answer)	• Continuous improvement is necessary to keep the language alive. Later this was not possible without disappointing users.
Affirmative (12 answers)	• Yes, we designed modular, reusable components, which became a major benefit of adoption. • Yes, we improved performance, reliability, and usability.
Not significantly (1 answer)	• I suspect those changes did not affect the language's success, except that if it had never made any progress, then it would have faded away.
Inconclusive (2 answers)	• Unclear if they helped with the success.
Language merits important (1 answer)	• Success mainly due to stability of the DSL and reuse of example DSL specifications (library).

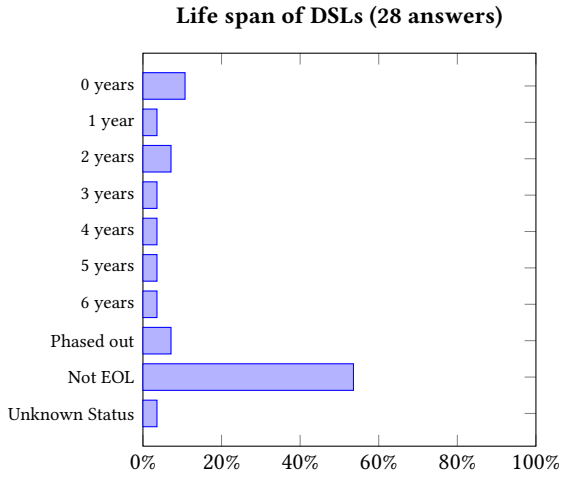

Figure 10: The life span of the surveyed DSLs. The findings stem from the answers to two different questions where some participants stated their DSL where *phased out* without mentioning a specific year.

breaking updates occur when DSL author has pragmatically estimated that the cost of migrating DSL programs is sufficiently low. We find this form of pragmatism to be a characteristic of DSLs as opposed to general-purpose languages where non-breaking updates, for the most part, are unthinkable.

3.5 End of Life

In investigating the end of life (EOL), we recognise that a) it can be complicated to say when a DSL is phased out and b) it is impossible to foresee when, why, and if a DSL will be phased out. Therefore, we asked participants to answer questions to their best ability even if their DSL was not at the EOL. We emphasise that these answers are a mixture of actual experiences and estimates.

Results. Many DSLs are long-term software projects that remain in use a decade after their introduction (Figure 10). Of the surveyed DSL, 53% of authors did not consider their DSL project to have reached its EOL. These DSLs have an average age of 12 years, and several of them estimate that they will remain in use for 5, 10, 15, or more years. (12) DSLs that had reached their EOL were commonly retired either due to changes in their domain, better tooling, or adoption by other languages (Figure 4). DSLs that were not phased out yet answered similarly for the expected reasons of their retirement with a focus on new languages or tooling replacing the DSL.

Finally, we asked survey participants whether there were any efforts that the DSL authors would have taken with the power of hindsight (Table 5). While this question does not investigate features of the surveyed DSL as such, we found it relevant to examine whether DSL creators had any prevalent lessons learned. Besides the authors who answered they would have made no efforts looking back, the answers highlight five different aspects. First, it is inevitable to have such efforts that they would have made. Second, creators should remember activities for attracting users, either through promotion or internal usage. Third, there are different ways of improving the development process. Such improvements are considered both from a process perspective and an architecture perspective. Fourth, the application domain can be difficult to work in, and creators can consider switching to other application domains. Fifth, case studies during development could have improved the language, but creators were unable to do so due to case studies

Table 4: What was, or will be, the primary reason for the DSL to have been phased out? (Best estimate if in the future) (21 answers)

Ecosystem changes (2 answers)	• Dependent on obsolete infrastructure. • Software base and applications moved on, and we lacked resources to keep the DSL updated.
Replaced by other language or tooling. (7 answers)	• Replaced by the SQL standard, making a standalone DSL somewhat redundant. • Replaced by new tooling • Better database design and centralised solutions mooted the problem being solved.
Domain changes (3 answers)	• Domain itself phased out. The language was reborn in new domains. • Important changes in the domain.
Not EOL (2 answers)	• This DSL is not EOL.
Lack of users (2 answers)	• Lack of continued commercial support, a need to rewrite the systems based on it, and lack of available expertise if usage declines.
Other (2 answers)	• Development team has moved to other projects.

Table 5: Looking back, are there any efforts that you would have taken during any of the previous phases? Why did you not take them at the time? (23 answers)

Usage and promotion (4 answers)	• We considered but ultimately chose not to follow to exchange hardcoded specifications for their DSL counterparts. However, the language and implementation are not sufficiently developed to support all specifications. • Promoting its use. This was not done because of the DSL's academic character and limited resources.
Usability (1 answer)	• Make it more user friendly: make the compilation more robust to failure for various corner cases; improve error reporting of syntactic/semantic errors.
Improve development (3 answers)	• Application context, domain application, and language evolved separately. It is possible that we could have unified them from the start with a bottom-up approach with nothing working end-to-end along the way. • Waited longer before committing to a DSL; rather, go for an opinionated library in a suitably flexible host language first. • We should have looked sooner for a fundamentally simpler architecture for the underlying system instead of assuming that it was fundamental.
Application domain (2 answers)	• The bottleneck for use was never the language itself but state of the art in the domain systems it targeted. • We could have focused more on a broader domain earlier, and we could have pushed on off the shelf reusable components earlier. We could have focused more on fault tolerance, including better support for lower performance applications.
Use case studies (2 answers)	• Developing larger examples in the language would have helped. Doing that while developing the language and preparing the paper as part of a large team was challenging at the time. • Consider more complex use cases during the DSL specification; impossible due to lack of human resources.
Inevitable (2 answers)	• In a language that develops incrementally, at certain points, one is likely to wish one had done some things differently, but they can't be redone on account of maintaining backward compatibility. • One always has a few regrets: features hastily added to the language early on, which now seem redundant or awkward, for example, or implementation decisions that made the compiler code-base harder to maintain decades later on. But basically, no, I wouldn't do much of it differently if offered the chance.
Negative (5 answers)	• No
Other (1 answer)	• While interesting, DSL development is not my primary research field.

being a labour-intensive endeavour.

Discussion. We find that while a DSL serves as an alternative to using a GPL, it may be so successful that it is adopted by GPLs or other tools. This adoption is explicitly mentioned by three respondents, and it is also mentioned implicitly by three others since they expect their DSL to become obsolete when more mainstream tools or languages adopt the functionality the DSL provides. That is, there is a dual movement of ideas between DSLs and in GPLs or tools in that DSLs are created due to inefficiency in GPLs or tools, which in turn may evolve seeing the utility demonstrated by the DSL.

While we identified (8) promotion to be an important aspect of launching a DSL, the retrospective answers indicate that promotion may be a somewhat neglected activity. We also consider what DSL authors did not mention as problems retrospectively. While we found that evolution is important for the success of many DSLs (11), only three answers relate directly handling evolution differently, not counting those that discuss looking to expand the DSLs' domain. Likewise, while four answers consider usability and use cases of the DSL, there are no explicit regrets as to how users were involved in the design process. (13) These findings indicate that current ways of having user involvement in the design process are not causing dissatisfaction among DSL creators.

4 RECOMMENDATIONS

Based on the insights gained in our survey, we have six recommendations for DSL practitioners, educators, and the field in general. In doing so, we seek a balance between, on the one hand, recognising the value of the established practices mentioned by DSL authors and, on the other hand, challenging these practices.

R1: Explore practices for expert evaluation of DSLs. We found that expert evaluation is a widespread evaluation technique used by 57% of respondents (2). From this result, we conjectured that the method of expert evaluation has a significant impact on the design of many DSLs (5). However, expert evaluation is susceptible to the profile of the evaluating expert and how the expert evaluates the language. While we also found that 22% of respondents use heuristics evaluation, there still is a noticeable gap in unexplored practices. Therefore, we recommend further investigations into practices on expert evaluation of DSLs with two purposes: First, academia should explore how expert evaluation is performed, by whom, using which artefacts, and to what effect. One should compare the effectiveness of using expert evaluation against comparable techniques. Second, due to the widespread use of the techniques, we find a need for academia and industry to establish guidelines for practitioners jointly. We hypothesise that the guidelines should

be low-cost and easily approachable if they are to be followed.

R2: Guidelines for the level and kind of user involvement. We find that DSL creators have vastly different approaches to the kind and level of user involvement during DSL design (6). The user involvement ranges from none at all, to requirement elicitation, to use-case design, to language evaluation, to users being part of the design team. Although we found no indication of dissatisfaction regarding user involvement (13), we still find a need for guidelines for how and when users should be involved in a specific design project. We hypothesised that there could be a correlation between the level of user involvement and the programming experience of users but found none (7). Still, we believe that the prescriptions should be based on the design context, such as who is the target user, how available are users, what kind of tasks they are to perform, and how complex the DSL is.

R3: Consider the purpose of heuristic design principles. We found that heuristic evaluation is used only by 22% of survey participants (3). Still, heuristic design principles are valuable knowledge since they are the expressed experience of experts within the field. We hypothesise that the utility of heuristic design principles could increase if they allow for more lightweight evaluation activities or can be applied generatively in the design process. Therefore, we recommend that the creators of heuristic design guidelines consider how and when these guidelines should be used in DSL design.

R4: Use the flexibility allowed by the project. We find that DSL authors report many different ways of being flexible or pragmatic in their approach to DSL design. To mention three examples: First, authors found it necessary to introduce breaking updates to their languages. One respondent reported that this was necessary to accommodate a shift in research focus to a different underlying technology. Second, authors found new application domains for their DSL project. One respondent reported that their DSL became more successful by widening the range of applicable application domains. Third, authors report major improvements made to the DSL and its launch. These improvements come in many forms, such as performance, usability, and modularity. From these insights, we recommend that DSL practitioners should use the flexibility allowed by their project as opposed to being dogmatic in their development approach.

R5: Consider evolution as an intrinsic part of DSL creation. We found that evolution is a part of most DSLs' life cycles and that this evolution is important for the success of many DSLs (9) (11). We have already mentioned some examples of the importance of evolution in recommendation *R4*. While this finding may not be surprising for many practitioners and scholars within the field, we find it essential to substantiate this claim empirically. From this finding, we have a three-fold recommendation. First, creators of DSLs should consider evolution as an aspect of DSL creation. Second, tool creators should continue developing tools with dedicated support for managing the evolution of language specifications and artefacts. Third, educators, methods, and textbooks on DSL design should treat evolution with utmost importance.

R6: Consider EOL through adoption a success criterion. We found that DSL authors report that their DSL reached (or will reach) EOL due to the DSL's functionality being adopted by tools or other languages (12). One example is a language that has become obsolete since its functionality was incorporated into a mainstream and widely used standard. While the EOL of a DSL through being absorbed into another language or tool is commonly viewed as defeat, we recommend declaring this as one potential success. Furthermore, we recommend using the means of language engineering for the DSL to influence standard concepts and constructs used to represent the domain.

5 THREATS TO VALIDITY

We have mitigated threats to validity originating from our sampling method and our way of conducting the questionnaire.

Internal Validity relates to the degree to which we can trust the findings within our survey. There is an internal validity risk of not measuring the intended phenomenon when conducting a questionnaire. We mitigated this risk of measuring something unintended in four ways. First, we presented and discussed both the questionnaire's purpose and questions with a colleague. Second, we conducted a blind pilot run of the study with another colleague and subsequently discussed their understanding of the asked questions. Third, for all answers that did not solely establish a fact, we allowed participants to submit free-text answers allowing them to answer a question differently than we had intended. Fourth, in our presentation, we are transparent in what interpretation we have made.

Another risk to internal validity is that there is insufficient data for the claims made in our findings. Our primary mitigation of this risk was to obtain a sample of sufficient size so as to be less susceptible to noise. Also, while we do not find any correlation, we find the sample size to be adequate for our chosen method. We found that including *extra* submissions did not change our statistical findings but did provide valuable examples.

We considered excluding DSLs introduced before 1990 to avoid too diverse subject DSLs since this might threaten internal validity. However, old DSLs should be included to increase internal validity since an exclusion criterion comes with the following three methodological problems. First, the survey seeks to investigate the entire life cycle of DSLs, meaning that older DSLs are relevant for the survey. Second, using the exclusion criterion, we would have presupposed the lifetime and life cycle of DSLs. Third, if an old DSL is still in use, then it is at least as interesting as a more modern one when examining the current state of DSLs' life cycle.

External Validity relates to the degree to which we should believe our findings to be transferable. From this perspective, the most severe threat is the application domain of the surveyed domain. We recognise that the chosen collections of DSLs to sample from are biased towards DSLs within finance and robotics domains. Therefore, our survey has the highest degree of external validity when generalising to other DSLs within these and similar domains.

Still, we have mitigated the influence of the bias on our findings by avoiding questions directly influenced by the domain, such as tooling, application context, general kinds of domain tasks, monetary costs, and underlying programming paradigms.

Another potential bias in the sample is that the survey sample does not represent the actual population of DSLs. We recognise that our sampling method is biased towards receiving answers from newer publications for a multitude of reasons, e.g., authors changing email addresses, authors retiring, DSL projects retiring, companies dissolving, or newer authors being more excited about their work being noticed. Therefore, we present the age to characterise the DSL sample to make the bias transparent. We had mitigation measures seeking a sample as representative as possible. First, we have sought multiple channels of contacting all invited authors. Second, to avoid participants getting stuck on non-applicable questions or questions participants could not answer, we allowed skipping questions. We deemed that older DSLs had a higher risk of encountering these kinds of problems. Third, to accommodate participants who did not want to submit data on the third-party platform, we allowed participants to email their answers.

6 RELATED WORK

Several studies seek to investigate the current state of different DSL topics through meta-studies of the field. These studies, which we describe below, use three different sources to obtain information. First, *zoo analysis* considers software itself as the primary source of information. Second, *publication analysis* considers publications as the primary source of information. Third, *questionnaire analysis* uses self-reports from questionnaire recipients as the primary source of information. Both zoo analysis and publication analysis come with strengths and weaknesses compared to the questionnaire approach. Zoo analysis does not rely on interpreting DSL authors' possibly biased reporting, but they cannot answer development-oriented research questions. Publication analysis may answer development-oriented research questions but relies on authors' prior reports on areas of interest and can only answer questions to the degree that report uniformity and granularity allows.

Using zoo analysis, Dragule et al. survey DSLs for robotic missions [15]. They identify and categorise 30 robotic mission programming environments and present their design space through a feature model of the environments. Similarly, Kapre and Bayliss [22] survey 9 DSLs used for high-performance FPGA computing. Schauss et al. create a chrestomathy of DSL implementations to teach implementation techniques [34]. They subsequently conduct a pure zoo analysis of their implementation to identify implementation variations of interest. For this purpose, several DSL zoos are open for future zoo analysis [8, 9].

Using publication analysis, Deursen et al. [39], Marnik et al. [28], and Oliveira et al. [31] all made early investigations into DSL development and implementation methodologies by reviewing selected publications. More recently, Nascimento et al. [29] and Kosar et al. [24] have conducted systematic mapping studies to investigate research within the field. Of relevance to our study, Kosar et al. find that much research proposes new techniques supporting different development phases with an overwhelming focus on design, implementation, and domain analysis, with very few considering maintenance and validation. Iung et al. [21] conducted a similar systematic mapping study for tools being used by DSL creators. On the same topic, Erdweg et al. [16] evaluate and compares language workbenches.

Systematic literature reviews are also used as a form of publication analysis. Like our paper, Poltronieri et al. investigate how DSL authors evaluate their DSLs' [33] and create a taxonomy for evaluation on this basis. In an updated review [32], they find usability evaluation to be the most often used evaluation technique. However, they also find that even after applying a quality assessment filter only, 13 out of 21 describe their used technique. This finding points to a methodological difficulty in examining some topics through publication analysis. As such, our method can be seen as a different angle of attack with its own threats to validity.

Questionnaire analyses have been used to explore the broader topic of model-based engineering. For example, Broy et al. [14] investigate the benefit of using model-based practices within the car industry, Badreddin et al. [11] investigate trends in software practitioners' use of model practices, and Liebel et al. [26] investigate students' perception of modelling tools and UML.

7 CONCLUSION

In this paper, we have presented a survey to establish current practices in managing the life cycle of DSLs through a questionnaire. The 30 answers from the authors of DSLs provide us with several findings relating to the DSL management phases of the design and development, launch, evolution, and end of life. Among others, we find that a) there is no established practice as to the level of user involvement during development, b) DSL authors find demonstrating the merits of a DSL is important during launch, c) handling evolution correctly is important for the success of a DSL, and d) that it is common for DSLs to be replaced by other tooling or languages. Based on our findings, we have presented six recommendations relating to different phases of a DSL's life cycle. Among others, we recommend a) further explorations of practices for expert evaluation, b) that DSL practitioners are flexible in their approach to DSL development, and c) that DSL evolution as a topic is treated with utmost importance both in education, industry, and academia. For future work, we propose that an open research database is created where DSL creators register and update information on their DSL. Such a database would provide the community with high-quality information and allow more process-oriented research but require a method for deciding what the database should contain.

8 ACKNOWLEDGEMENT

We thank Innovation Fund Denmark (7076-00029B) for funding, all questionnaire recipients for their time and contributions, and Peter Sestoft for envisioning and supervising the work.

REFERENCES

[1] 1999. DSL '99: Proceedings of the 2nd Conference on Domain-Specific Languages. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/331960

[2] 2013. FPCDSL '13: Proceedings of the 1st Annual Workshop on Functional Programming Concepts in Domain-Specific Languages. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2505351

[3] 2016. RWDSL '16: Proceedings of the 1st International Workshop on Real World Domain Specific Languages. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2889420

[4] 2017. RWDSL17: Proceedings of the 2nd International Workshop on Real World Domain Specific Languages. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3039895

[5] 2018. Category:Domain-specific programming languages. https://en.wikipedia.org/w/index.php?title=Category:Domain-specific_programming_languages&oldid=858848463 accessed 1 May 2021.

[6] 2018. RWDSL2018: Proceedings of the Real World Domain Specific Languages Workshop 2018. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3183895

[7] 2019. RWDSL '19: Proceedings of the 4th ACM International Workshop on Real World Domain Specific Languages. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3300111

[8] 2020. DevBoost/EMFText-Zoo. https://github.com/DevBoost/EMFText-Zoo original-date: 2012-08-07T07:22:54Z.

[9] 2020. Index – Robotics DSL Zoo. https://corlab.github.io/dslzoo/all.html accessed 1 May 2021.

[10] 2021. Financial Domain-Specific Language Listing and Resources. http://www.dslfin.org/resources.html accessed 1 May 2021.

[11] Omar Badreddin, Rahad Khandoker, Andrew Forward, Omar Masmali, and Timothy C. Lethbridge. 2018. A Decade of Software Design and Modeling: A Survey to Uncover Trends of the Practice. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, Copenhagen Denmark, 245–255. https://doi.org/10.1145/3239372.3239389

[12] Ankica Barišić, João Cambeiro, Vasco Amaral, Miguel Goulão, and Tarquínio Mota. 2018. Leveraging teenagers feedback in the development of a domain-specific language: the case of programming low-cost robots. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, Pau France, 1221–1229. https://doi.org/10.1145/3167132.3167264

[13] Holger Borum. 2022. Artefact: Survey of Established Practices in the Life Cycle of Domain-Specific Languages. https://github.com/hborum/models-22-survey original-date: 2022-07-22T06:26:06Z.

[14] Manfred Broy, Sascha Kirstan, Helmut Krcmar, and Bernhard Schätz. 2012. What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry? https://doi.org/10.4018/978-1-61350-438-3.ch013 ISBN: 9781613504383 Library Catalog: www.igi-global.com Pages: 343-369 Publisher: IGI Global.

[15] Swaib Dragule, Sergio García Gonzalo, Thorsten Berger, and Patrizio Pelliccione. 2021. Languages for Specifying Missions of Robotic Applications. In *Software Engineering for Robotics*, Ana Cavalcanti, Brijesh Dongol, Rob Hierons, Jon Timmis, and Jim Woodcock (Eds.). Springer International Publishing, Cham, 377–411. https://doi.org/10.1007/978-3-030-66494-7_12

[16] Sebastian Erdweg, Tijs van der Storm, Markus Völter, Laurence Tratt, Remi Bosman, William R. Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, Gabriël Konat, Pedro J. Molina, Martin Palatnik, Risto Pohjonen, Eugen Schindler, Klemens Schindler, Riccardo Solmi, Vlad Vergu, Eelco Visser, Kevin van der Vlist, Guido Wachsmuth, and Jimi van der Woning. 2015. Evaluating and comparing language workbenches: Existing results and benchmarks for the future. *Computer Languages, Systems & Structures* 44 (Dec. 2015), 24–47. https://doi.org/10.1016/j.cl.2015.08.007

[17] Marco Frigerio, Jonas Buchli, and Darwin G. Caldwell. 2013. A Domain Specific Language for kinematic models and fast implementations of robot dynamics algorithms. *arXiv:1301.7190 [cs]* (Jan. 2013). http://arxiv.org/abs/1301.7190 arXiv: 1301.7190.

[18] Ruediger Gad. 2017. Evolution of a Stream Transformation DSL. In *Proceedings of the 2nd International Workshop on Real World Domain Specific Languages - RWDSL17*. ACM Press, Austin, TX, USA, 1–10. https://doi.org/10.1145/3039895.3039897

[19] Felienne Hermans, Martin Pinzger, and Arie van Deursen. 2009. Domain-Specific Languages in Practice: A User Study on the Success Factors. In *Model Driven Engineering Languages and Systems (Lecture Notes in Computer Science)*, Andy Schürr and Bran Selic (Eds.). Springer, Berlin, Heidelberg, 423–437. https://doi.org/10.1007/978-3-642-04425-0_33

[20] Nico Hochgeschwender, Sven Schneider, Holger Voos, and Gerhard K Kraetzschmar. 2014. Declarative Specification of Robot Perception Architectures. (2014), 12.

[21] Aníbal Iung, João Carbonell, Luciano Marchezan, Elder Rodrigues, Maicon Bernardino, Fabio Paulo Basso, and Bruno Medeiros. 2020. Systematic mapping study on domain-specific language development tools. *Empirical Software Engineering* 25, 5 (Sept. 2020), 4205–4249. https://doi.org/10.1007/s10664-020-09872-1

[22] Nachiket Kapre and Samuel Bayliss. 2016. Survey of domain-specific languages for FPGA computing. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. 1–12. https://doi.org/10.1109/FPL.2016.7577380 ISSN: 1946-1488.

[23] Mika Karaila. 2009. Evolution of a Domain Specific Language and its engineering environment - Lehman's laws revisited. (2009), 7.

[24] Tomaž Kosar, Sudev Bohra, and Marjan Mernik. 2016. Domain-Specific Languages: A Systematic Mapping Study. *Information and Software Technology* 71 (March 2016), 77–91. https://doi.org/10.1016/j.infsof.2015.11.001

[25] J.R. Lewis and B. Martin. 2003. Cryptol: high assurance, retargetable crypto development and validation. In *IEEE Military Communications Conference, 2003. MILCOM 2003.*, Vol. 2. 820–825 Vol.2. https://doi.org/10.1109/MILCOM.2003.1290218

[26] Grischa Liebel, Omar Badreddin, and Rogardt Heldal. 2017. Model Driven Software Engineering in Education: A Multi-Case Study on Perception of Tools and UML. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)*. 124–133. https://doi.org/10.1109/CSEET.2017.29 ISSN: 2377-570X.

[27] Sandra Macià, Sergi Mateo, Pedro J. Martínez-Ferrer, Vicenç Beltran, Daniel Mira, and Eduard Ayguadé. 2018. Saiph: Towards a DSL for High-Performance Computational Fluid Dynamics. In *Proceedings of the Real World Domain Specific Languages Workshop 2018 (RWDSL2018)*. Association for Computing Machinery, Vienna, Austria, 1–10. https://doi.org/10.1145/3183895.3183896

[28] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *Comput. Surveys* 37, 4 (Dec. 2005), 316–344. https://doi.org/10.1145/1118890.1118892

[29] Leandro Nascimento, Daniel Viana, Paulo Neto, Dhiego Martins, Vinicius Garcia, and Silvio Meira. 2012. A Systematic Mapping Study on Domain-Specific Languages.

[30] Arne Nordmann, Nico Hochgeschwender, and Sebastian Wrede. 2014. A Survey on Domain-Specific Languages in Robotics. In *Simulation, Modeling, and Programming for Autonomous Robots (Lecture Notes in Computer Science)*, Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald (Eds.). Springer International Publishing, Cham, 195–206. https://doi.org/10.1007/978-3-319-11900-7_17

[31] Nuno Oliveira, Maria João Pereira, Pedro Rangel Henriques, and Daniela Cruz. 2009. Domain specific languages: a theoretical survey. *INForum'09 - Simpósio de Informática* (2009). https://bibliotecadigital.ipb.pt/handle/10198/1192 Accepted: 2009-10-01T12:52:37Z Publisher: Faculdade de Ciências da Universidade de Lisboa.

[32] Ildevana Poltronieri, Allan Christopher Pedroso, Avelino Francisco Zorzo, Maicon Bernardino, and Marcia de Borba Campos. 2021. Is Usability Evaluation of DSL Still a Trending Topic? In *Human-Computer Interaction. Theory, Methods and Tools*, Masaaki Kurosu (Ed.). Vol. 12762. Springer International Publishing, Cham, 299–317. https://doi.org/10.1007/978-3-030-78462-1_23 Series Title: Lecture Notes in Computer Science.

[33] Ildevana Poltronieri, Avelino Francisco Zorzo, Maicon Bernardino, and Marcia de Borba Campos. 2018. Usa-DSL: usability evaluation framework for domain-specific languages. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18)*. Association for Computing Machinery, Pau, France, 2013–2021. https://doi.org/10.1145/3167132.3167348

[34] Simon Schauss, Ralf Lämmel, Johannes Härtel, Marcel Heinz, Kevin Klein, Lukas Härtel, and Thorsten Berger. 2017. A chrestomathy of DSL implementations. In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering (SLE 2017)*. Association for Computing Machinery, Vancouver, BC, Canada, 103–114. https://doi.org/10.1145/3136014.3136038

[35] Mathijs Schuts, Marco Alonso, and Jozef Hooman. 2021. Industrial experiences with the evolution of a DSL. In *Proceedings of the 18th ACM SIGPLAN International Workshop on Domain-Specific Modeling*. Association for Computing Machinery, New York, NY, USA, 21–30. https://doi.org/10.1145/3486603.3486774

[36] Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. 2018. Q#: Enabling scalable quantum computing and development with a high-level domain-specific language. *Proceedings of the Real World Domain Specific Languages Workshop 2018 on - RWDSL2018* (2018), 1–10. https://doi.org/10.1145/3183895.3183901 arXiv: 1803.00652.

[37] Marcel van Amstel, Mark van den Brand, and Luc Engelen. 2010. An exercise in iterative domain-specific language design?. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE) on - IWPSE-EVOL '10*. ACM Press, Antwerp, Belgium, 48. https://doi.org/10.1145/1862372.1862386

[38] Tijs van der Storm and Sebastian Erdweg. 2015. Proceedings of the 3rd Workshop on Domain-Specific Language Design and Implementation (DSLDI 2015). *arXiv:1508.03536 [cs]* (Aug. 2015). http://arxiv.org/abs/1508.03536 arXiv: 1508.03536.

[39] Arie van Deursen, Paul Klint, and Joost Visser. 2000. Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices* 35, 6 (June 2000), 26–36.

https://doi.org/10.1145/352029.352035

[40] Markus Voelter, Bernd Kolb, Klaus Birken, Federico Tomassetti, Patrick Alff, Laurent Wiart, Andreas Wortmann, and Arne Nordmann. 2019. Using language workbenches and domain-specific languages for safety-critical software development. *Software & Systems Modeling* 18, 4 (Aug. 2019), 2507–2530. https://doi.org/10.1007/s10270-018-0679-0

[41] David Wile. 2004. Lessons learned from real DSL experiments. *Science of Computer Programming* 51, 3 (June 2004), 265–290. https://doi.org/10.1016/j.scico.2003.12.006