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

Holger Stadel Borum  
hsth@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:

1 INTRODUCTION
Domain-specific languages (DSLs) have been established as a multifaceted 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.
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 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, dsfin.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)
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. 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, 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, 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. 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.
Which methods were used to evaluate the usability of the DSL during design and development? (30 responses)

- Usability tests
- User workshops
- Heuristic evaluation
- Expert evaluation
- Case studies
- Actual usage

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.

Different human-centred evaluation methods\(^3\) are used by 53% of DSLs. Finally, \(\ominus\) 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). \(\ominus\) 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. \(\ominus\) 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 \(\ominus\). 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, \(\ominus\) 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.

\[^3\text{This category consists of usability tests and user workshops.}\]
Table 1: How were users involved in the language design? (30 responses)

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

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.

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 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. 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)

<table>
<thead>
<tr>
<th>Affirmative (9 answers)</th>
<th>Language merits important (7 answers)</th>
<th>Other (7 answers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Yes, but some users prefer to stick to the GUI.</td>
<td>• 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.</td>
<td>• Users had by policy to use the DSL.</td>
</tr>
<tr>
<td>• Yes. I don’t think people would have adopted it if we hadn’t actively promoted and sold it.</td>
<td>• Yes, the transition from earlier languages was effective.</td>
<td>• 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.</td>
</tr>
<tr>
<td>• Yes, the transition from earlier languages was effective.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What factors have contributed to evolving the DSL after its launch? (29 answers)

- Changes in implementation technologies
- Changes in internal technologies
- Changes in external technologies
- New user wishes
- An agile development process
- New areas of application
- Changes in the existing application domain
- Merger with other DSL
- Little evolution or no evolution reported
- Major improvements of the language implementation
- Expanding the application domain of the DSL
- Major revisions of the learning materials
- Major improvements of the development environment
- Major revisions of the documentation
- Major changes to the general language syntax
- Breaking updates
- Transition to other DSL

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 major evolution of the ecosystem of DSLs also occurs but is rarer.
Table 3: Do you think these efforts affected the success of the DSL? Why? (21 answers)

<table>
<thead>
<tr>
<th>Category</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vital</td>
<td>• Continuous improvement is necessary to keep the language alive. Later this was not possible without disappointing users.</td>
</tr>
<tr>
<td>Affirmative</td>
<td>• Yes, we designed modular, reusable components, which became a major benefit of adoption.</td>
</tr>
<tr>
<td></td>
<td>• Yes, we improved performance, reliability, and usability.</td>
</tr>
<tr>
<td>Not significantly</td>
<td>• 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.</td>
</tr>
<tr>
<td>Inconclusive</td>
<td>• Unclear if they helped with the success.</td>
</tr>
<tr>
<td>Language merits</td>
<td>• Success mainly due to stability of the DSL and reuse of example DSL specifications (library).</td>
</tr>
</tbody>
</table>

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. (3) 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.

![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.](image-url)
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 | • 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.  
| (4 answers)          | • Promoting its use. This was not done because of the DSL’s academic character and limited resources. |
| Usability            | • Make it more user friendly; make the compilation more robust to failure for various corner cases; improve error reporting of syntactic/semantic errors.  
| (1 answer)           | |
| Improve development  | • 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.  
| (3 answers)          | • 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   | • The bottleneck for use was never the language itself but state of the art in the domain systems it targeted.  
| (2 answers)          | |
| Use case studies     | • 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.  
| (2 answers)          | • Consider more complex use cases during the DSL specification; impossible due to lack of human resources. |
| Inevitable           | • 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.  
| (2 answers)          | • 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             | • No  
| (5 answers)          | |
| Other                | • While interesting, DSL development is not my primary research field.  
| (1 answer)           | |

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 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 promotion is important for the success of many DSLs, only three answers relate directly handling evolution differently, not counting those that discuss looking to expand the DSL’s 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. 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. From this result, we conjectured that the method of expert evaluation has a significant impact on the design of many DSLs. 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. 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, 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. 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. 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. 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. 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. Jung 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 tools 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.