Usability Heuristics for Domain-Specific Languages (DSLs)

Eduardo Mosqueira-Rey
University of A Coruña
A Coruña, Spain
eduardo@udc.es

David Alonso-Ríos
University of A Coruña
A Coruña, Spain
dalonso@udc.es

ABSTRACT

The usability of Domain-Specific Languages (DSLs) has been attracting considerable interest from researchers lately. In particular, our literature review found many usability studies that make use of subjective and empirical methods. However, we noted a lack of heuristic methods in the literature. In comparison, there exist several usability studies of Application Programming Interfaces (APIs) that have used heuristics with success, so we argue that this approach would be also useful for DSLs. Therefore, this paper proposes a set of usability heuristics for DSLs and illustrates the approach with a case study. We show how our heuristics helped us identify many actual usability problems, even for a simple DSL.

CCS CONCEPTS

• **Human-centered computing → HCI theory, concepts and models**; • **Software and its engineering → Designing software**.

KEYWORDS

Domain-Specific Languages, DSL, Heuristic Evaluation, Usability, Heuristics, Usability Taxonomy

ACM Reference Format:
Eduardo Mosqueira-Rey and David Alonso-Ríos. 2020. Usability Heuristics for Domain-Specific Languages (DSLs). In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3341105.3374234

1 INTRODUCTION

A Domain-Specific Language (DSL) is expressly aimed at a particular domain and its expressiveness is typically limited. DSLs are usually classified as either external or internal [5]. An **external DSL** is a language that is entirely independent from the core language used in the application. An **internal DSL** is, in fact, a subset of the core language that has very specific goals and offers a restricted set of functionalities.

It could be argued that the distinction between an internal DSL and a typical API (Application Programming Interface) based on commands and queries is somewhat fuzzy. According to Fowler [5], the difference lies in the nature of the language. Command-query APIs specify a vocabulary for a domain, while internal DSLs also include an internal grammar. While API methods usually stand on their own, so to speak, internal DSL methods are often better understood as part of a larger expression. Internal DSLs lead us to the concept of *fluent interfaces*, which use method chaining to make the syntax closer to natural language.

Since there is more literature on API usability than on DSL usability, we decided to use the previously existing research on API usability as a springboard to identify usability heuristics for DSLs. Moreover, we aimed to follow a systematic methodology that explicitly connected our heuristics to established usability criteria from the literature.

2 BACKGROUND

A great number of methods exist for studying usability in general, and the same is true for the more specialized field of DSL usability. We conducted a literature review of DSL usability studies and found that many kinds of traditional usability methods are typically employed, such as user tests, performance measurements, surveys, questionnaires, interviews, and so on. This diversity of techniques is summarized in the Usa-DSL Framework for DSL usability [10]. There is also a noticeable lack of consensus regarding which usability definition (e.g., ISO 9241-11, Jakob Nielsen's, some subset of these, etc.) is to be followed in order to establish usability criteria, metrics, and measures. Moreover, we could not find any actual studies that made use of expert methods like heuristic evaluation or guideline review. Heuristic evaluation can be defined as "expert identifies violations of heuristics", and guideline review as "expert checks guideline conformance" [7].

As far as we know, no actual lists of usability heuristics or guidelines have ever been proposed for the field of DSLs. This contrasts with usability studies of computer applications, where expert methods are extensively used and popular checklists of guidelines and heuristics exist. One possible explanation for this could be that "the Usability evaluation of a UI is typically superficial when compared to the required usability evaluation of DSLs" [4, p. 345]. This means that DSLs require more specific usability heuristics and guidelines.

3 METHODOLOGY

Our work is based in the usability taxonomy proposed by Alonso-Ríos et al. [2]. This model aims to synthesize the existing definitions of usability and organize their attributes into a hierarchy. Figure 1 shows the usability taxonomy in a simplified version with, at the highest level, the attributes of *Knowability*, *Operability*, *Efficiency*, *Robustness*, *Safety* and *Subjective Satisfaction*.

This usability taxonomy has been further used by these authors as the basis of a more systematic and generalizable approach to heuristic evaluation [1]. The rationale is as follows: Heuristic evaluation has traditionally been a quick and cheap usability technique, based on a generic set of rules of thumb, such as Jakob Nielsen's

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03.
https://doi.org/10.1145/3341105.3374234

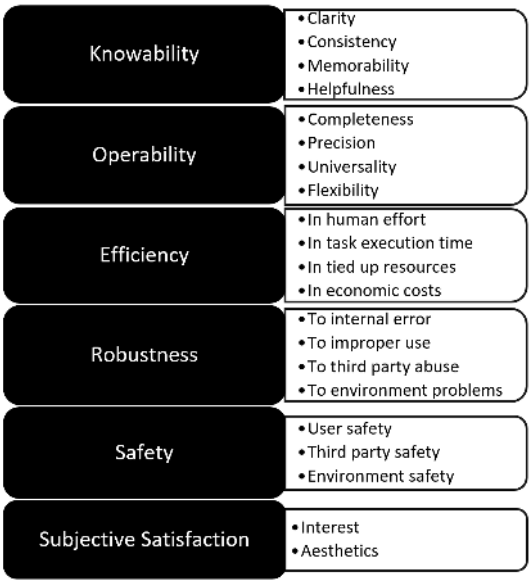

Figure 1: Usability taxonomy

heuristics [9]. This is enough in some contexts but, in others, heuristics are inevitably complemented by ad hoc judgments or deep knowledge of guidelines. In the field of DSL usability in particular, researchers such as Barišić et al., found that heuristic evaluation is "often regarded as not being capable to encompass all usability attributes" [3, p. 123]. This is why we argue that a more comprehensive and systematic view of usability is needed here.

After reviewing the literature, we could not find any heuristics expressly made for DSLs. However, DSLs and APIs are closely related, so we used the usability heuristics for APIs proposed by Mosqueira-Rey et al. [8] to create our own. This was done taking into account the differences and similarities between APIs and DSLs. Finally, we mapped the resulting usability heuristics for DSLs into the usability taxonomy. The goal of this step was to validate the former and try to identify unaddressed usability aspects.

4 PROPOSED DSL HEURISTICS

Tables 1 to 9 show our proposed heuristics for DSL usability. As mentioned above, they are derived and adapted from the heuristics for API usability by Mosqueira-Rey et al. [8].

5 CASE STUDY

We tested our heuristics with a case study. We chose the IEC 61131-7 standard [6] that defines a simple DSL for the programming of fuzzy control applications used by programmable controllers. This DSL includes the following elements:

- **Function block**: A block that represents functions and follows the definition previously given in the standard IEC 61131-3.
- **Input and output parameters**: Parameters that are passed into and out of the function block. The data types of these parameters shall be defined according to IEC 61131-3.

Table 1: Heuristics for Knowability (part 1: Clarity)

Code	Heuristics
K-1	Elements (keywords, operators, etc.) should be self-explanatory.
K-2	When reading code, it should be easy to understand what that code does.
K-3	The elements of the DSL should be loosely coupled.
K-4	Elements should focus on doing one thing.
K-5	When writing code it should be easy to know what elements of the DSL to use.
K-6	When writing code it should be easy to know where you are at any point.

Table 2: Heuristics for Knowability (part 2: Consistency)

Code	Heuristics
K-7	The syntax for the DSL elements should be self-consistent.
K-8	The semantics of the DSL elements should be self-consistent.
K-9	The syntax of the terminology used for the DSL elements should be consistent with standard conventions.
K-10	The semantics of the terminology used for the DSL elements should be consistent with standard conventions.
K-11	The syntax of the structures of the DSL should be self-consistent.
K-12	The semantics of the structures of the DSL should be self-consistent.

Table 3: Heuristics for Knowability (part 3: Memorability)

Code	Heuristics
K-13	The syntax of DSL elements should be easy to remember.
K-14	The semantics of DSL elements should be easy to remember.
K-15	The syntax of DSL structures should be easy to remember.
K-16	The semantics of DSL structures should be easy to remember.

Table 4: Heuristics for Knowability (part 4: Helpfulness)

Code	Heuristics
K-17	Every DSL element should be documented.
K-18	Documentation and comments should not include inappropriate information.
K-19	The DSL should properly identify deprecated elements (if any).
K-20	The DSL should provide helpful information in case of error and suggest solutions (if possible).
K-21	The documentation should include code samples for the most common scenarios.

Table 5: Heuristics for Operability

Code	Heuristic
O-1	The DSL should provide the necessary functionalities for domain tasks.
O-2	Data types should be as precise as necessary.
O-3	The DSL should avoid using elements (units, formats, spellings, etc.) that are not universally recognized.
O-4	The DSL should be open to extension.
O-5	The DSL should be flexible enough to allow changing code segments rather than whole modules.

Table 6: Heuristics for Efficiency

Code	Heuristics
E-1	The level of abstraction of the DSL should be adequate for the users and the domain.
E-2	The DSL should require as little typing as possible.
E-3	For complex DSLs, establish layers of complexity based on expertise.
E-4	The DSL should allow writing code that is efficient in terms of execution time.
E-5	The DSL should allow writing code that is efficient in terms of occupied resources.
E-6	The economic costs derived from using the DSL (if any) should be reasonable.

Table 7: Heuristics for Robustness

Code	Heuristics
R-1	The DSL should not have bugs in its functioning.
R-2	Runtime errors should be detected and, if possible, recovered from.
R-3	The DSL should not expose error-prone vulnerabilities.

Table 8: Heuristics for Safety

Code	Heuristics
S-1	The DSL should not get the user in legal trouble.
S-2	The DSL should clearly state its license of use.
S-3	The DSL should not compromise the confidentiality of users' personal information.
S-4	The DSL should not compromise the safety of users' assets.

Table 9: Heuristics for Subjective Satisfaction

Code	Heuristics
SS-1	Using the DSL should be satisfying.

- **Fuzzy variable**: Describes a fuzzy concept that has a name (e.g. *age*) and is composed of a series of fuzzy terms.
- **Fuzzy term**: Describes a particular fuzzy concept for a fuzzy variable (e.g. *old* and *young* are fuzzy terms in the *age* fuzzy variable). Each fuzzy term has an associated fuzzy set.
- **Fuzzy set**: A fuzzy set is a mapping of a set of numbers onto membership values that lie in the range [0, 1].
- **Fuzzification process**: The conversion of a numerical value into some degrees of membership of some fuzzy terms defined in a fuzzy variable.
- **Defuzzification process**: The conversion of some fuzzy terms and their corresponding membership degrees into a numerical value.
- **Fuzzy rule**: A typical IF-THEN rule in which the condition, the conclusion, or both, are fuzzy variables.
- **Fuzzy rule block**: A block comprising several fuzzy rules.
- **Weighting factor**: A value in the range [0, 1] that states the degree of importance, credibility or confidence of a fuzzy rule.

In Figure 2 we can see a code example of a fuzzy function block that includes all the elements defined above. In this case a fuzzy water controller is being simulated. We have two input variables that represent the temperature and the pressure of the water and one output variable that defines the position of a valve. Finally, the rules used by the controller are included, written with fuzzy terms and that are easy for humans to read and understand.

6 HEURISTIC EVALUATION OF THE CASE STUDY

We conducted a heuristic evaluation of the DSL using our own heuristics, and identified the following usability problems:

- **Some identifiers have no clear meaning**. For example, the defuzzification method uses constants to define a type of defuzzification: `CoG` (Centre of Gravity), `CoGS` (Centre of Gravity for Singletons), `CoA` (Centre of Area), `LM` (Left Most Maximum) and `RM` (Right Most Maximum). Some of these acronyms have no obvious meaning. In other words, some elements are not self-explanatory, which fails to meet heuristic K-1 from Table 1.
- **Some identifiers are difficult to remember**. Usability problems are often interrelated so, for example, the lack of clarity mentioned above also makes the DSL elements difficult to remember (heuristic K-14).
- **Lack of consistency in the identifiers**. Similarly, there is a lack of consistency in letter case and abbreviations. For example, `CoGS` (Centre of Gravity for Singletons) mixes upper case and lower case, and abbreviates the "of" but omits the "f" in "for" (heuristic K-13).
- **Use of verbs to represent entities**. When programming, we use verbs to represent actions, and nouns (or noun phrases) to represent objects and attributes. The IEC standard uses verbs like *FUZZIFY* to define fuzzy variables, but it would be better to use a noun, such as *FUZZY_VAR*. This is again a problem of clarity (heuristic K-1), and a problem of consistency (heuristic K-9) in that the semantics are not consistent with standard conventions.

```
FUNCTION_BLOCK Fuzzy_FB
VAR_INPUT
    temp:     REAL;
    pressure: REAL;
END_VAR
VAR_OUTPUT
    valve:    REAL;
END_VAR
FUZZIFY temp
    TERM cold     := (3, 1) (27, 0);
    TERM hot      := (3, 0) (27, 1);
END_FUZZIFY
FUZZIFY temp
    TERM low      := (55, 1) (95, 0);
    TERM high     := (55, 0) (95, 1);
END_FUZZIFY
DEFUZZIFY temp
    TERM drainage := -100;
    TERM closed   := 0;
    TERM inlet    := 100;
    METHOD: CoGS;
    DEFAULT       := 0;
END_DEFUZZIFY
RULEBLOCK No1
    AND: MIN;
    ACCU: MAX;
    RULE 1: IF temp IS cold AND pressure IS low
                            THEN valve IS inlet;
    RULE 2: IF temp IS cold AND pressure IS high
                            THEN valve IS closed WITH 0.8;
    RULE 3: IF temp IS hot  AND pressure IS low
                            THEN valve IS closed;
    RULE 4: IF temp IS hot  AND pressure IS high
                            THEN valve IS drainage;
END_RULEBLOCK
END_FUNCTION_BLOCK
```

Figure 2: Example of a fuzzy function block taken from IEC61131-7

- **Use of the same keyword to do two different things**. The *DEFUZZIFY* keyword is declaring a fuzzy variable but is also defining the defuzzify process over that variable. Conflating use and declaration fails to meet heuristic K-4, as elements should focus on doing one thing.
- **Use of different keywords to represent the same elements**. As a consequence of conflating the declaration of fuzzy variables and the fuzzify and defuzzify processes we found another usability problem: the IEC standard uses two different keywords (*FUZZIFY* and *DEFUZZIFY*) to declare fuzzy variables. Having completely opposite keywords for overlapping semantics is a problem of consistency (K-8).
- **The fuzzy terms can only use linear fuzzy sets**. In the standard, a membership function is defined as a piece-wise linear function by means of a table of points. It is not allowed to use non-linear functions such as Gaussian functions, which are typically used to define fuzzy sets. This is a problem of completeness, in that the DSL does not provide all the necessary functionalities (heuristic O-1).
- **There is only one fuzzification method**. We can choose between some defuzzification methods but there is only one way to fuzzify numerical values – which is implicitly the Mandani method – leaving out other methods such as the Larsen method. This is the same problem as above (O-1).

7 CONCLUSIONS

The main contribution of this paper is to propose new heuristics for analyzing DSL usability. In our literature review we found several works that include subjective and empirical methods of analysis, but we could not find any studies that made use of expert methods like heuristic evaluation or guideline review.

Since DSLs are very similar to APIs, especially the internal DSLs that some authors identify as *fluent* APIs, we can use API heuristics as a basis for developing DSL heuristics. Thus, the API heuristics proposed by Mosqueira-Rey et al. [8] are connected with an extended model of usability [2], which is something that helps us check if we are addressing actual usability problems.

We use the extended model of usability and the API heuristics to derive our own DSL heuristics. In order to test if the heuristics can identify usability problems in DSLs (whether internal of external) we choose a case study of an external DSL and apply the developed heuristics to it.

In the case study we were able to identify several usability problems during the heuristic evaluation even though the case study was a simple and relatively straightforward DSL. This means that our heuristics helped us find some usability problems or issues in the definition of DSLs and are a promising approach to expert usability evaluation for DSLs.

Moreover, since our heuristics are explicitly connected to a comprehensive usability model, we can categorize each usability issue into a particular usability criterion (clarity, consistency, etc.). This is something that helps to better identify the problems we are dealing with and how to solve them.

As a future work, we plan to refine the heuristics into detailed guidelines to perform also a guideline review. The idea is to apply heuristics and guidelines to both external and internal DSLs.

ACKNOWLEDGMENTS

This work has been supported partially by Xunta de Galicia (Spain) under projects GRC2014/035 and ED431G/01.

REFERENCES

[1] David Alonso-Ríos, Eduardo Mosqueira-Rey, and Vicente Moret-Bonillo. 2018. A Systematic and Generalizable Approach to the Heuristic Evaluation of User Interfaces. *International Journal of Human–Computer Interaction* 34, 12 (2018), 1169–1182. https://doi.org/10.1080/10447318.2018.1424101

[2] David Alonso-Ríos, Ana Vázquez-García, Eduardo Mosqueira-Rey, and Vicente Moret-Bonillo. 2009. Usability: a critical analysis and a taxonomy. *International Journal of Human-Computer Interaction* 26, 1 (2009), 53–74.

[3] Ankica Barišić, Vasco Amaral, and Miguel Goulão. 2018. Usability driven DSL development with USE-ME. *Comp. Lang., Systems & Structures* 51 (2018), 118–157.

[4] Ankica Barišić, Vasco Amaral, and Miguel Goulao. 2012. Usability Evaluation of Domain-Specific Languages. In *2012 8th Int. Conf. on the Quality of Information and Comm. Technology*. 342–347. https://doi.org/10.1109/QUATIC.2012.63

[5] Martin Fowler. 2010. *Domain-specific languages.* Pearson Education.

[6] IEC 61131-7 2000. *Programmable Controllers - Part 7: Fuzzy Control Programming.* Standard. International Electrotechnical Commission (IEC), Geneva, CH.

[7] Melody Y Ivory and Marti A Hearst. 2001. The state of the art in automating usability evaluation of user interfaces. *Comput. Surveys* 33, 4 (2001), 470–516.

[8] Eduardo Mosqueira-Rey, David Alonso-Ríos, Vicente Moret-Bonillo, Isaac Fernández-Varela, and Diego Álvarez-Estévez. 2018. A systematic approach to API usability: Taxonomy-derived criteria and a case study. *Information and Software Technology* 97 (2018), 46 – 63. https://doi.org/10.1016/j.infsof.2017.12.010

[9] Jakob Nielsen. 1995. 10 usability heuristics for user interface design. https://www.nngroup.com/articles/ten-usability-heuristics/. (1995). Acc.: 2018-09-01.

[10] Ildevana Poltronieri, Avelino Francisco Zorzo, Maicon Bernardino, and Marcia de Borba Campos. 2018. Usa-DSL: usability evaluation framework for domain-specific languages. In *33rd Annual ACM Sym. on App. Comp.* ACM, 2013–2021.