applied sciences

MDPI

Article

Experimental Comparison of Editor Types for Domain-Specific Languages

Sergej Chodarev *, Matúš Sulír, Jaroslav Porubän and Martina Kopčáková

Department of Computers and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
* Correspondence: sergej.chodarev@tuke.sk

Abstract: The editor type can influence the user experience for a domain-specific language, but empirical evaluation of this factor is still quite limited. In this paper, we present the results of our empirical study, in which we compare the productivity of users with different kinds of editors for the same domain-specific language. We chose the domain of quiz definitions and used three editors: a text editor with syntax highlighting and code completion developed with the Xtext framework, a projectional editor created using JetBrains MPS, and an existing form-based editor—Google Forms. The study was performed on 37 graduate students of computer science. The measured time was lower for the text editor than for the form-based editor, and the form-based editor's time was lower than the projectional one's; however, the results were statistically insignificant. The experiment was also complemented with a survey providing insight into the perception of different editor types by users.

Keywords: domain-specific languages; text editor; projectional editor; form-based editing; experiment

Citation: Chodarev, S.; Sulír, M.; Porubän, J.; Kopčáková, M. Experimental Comparison of Editor Types for Domain-Specific Languages. *Appl. Sci.* **2022**, *12*, 9893. https://doi.org/10.3390/app12199893

Academic Editor: Vito Conforti

Received: 29 July 2022
Accepted: 28 September 2022
Published: 1 October 2022

1. Introduction

A metamodel of a domain-specific language (DSL) defines its core concepts and structure, but user perception of a language is greatly influenced by the notation and the tools used to create and edit models expressed in the DSL [1]. There are not only purely textual vs. purely graphical languages and corresponding editors. Although the boundaries might not be precise, from the user's point of view, we can distinguish three types of DSL editors that combine textual and visual representation differently and provide different modes of interaction: advanced textual editors, projectional editors, and form-based editors.

Advanced textual editors or IDEs (integrated development environments) enhance the text with syntax highlighting or even display some additional visual information [2,3]. An advanced editor can also help with entering or modifying the text, using such features as code completion or refactoring operations. Advanced editors provide suggestions to the user on what can be entered in a specific context and what is the meaning of each alternative. This allows users to discover language constructs directly during the programming process or helps them to recall the exact names and syntax of language constructs.

Another option is a projectional editor [4] that allows users to modify the internal representation of a model using editable projection and therefore eliminates parsing. The projection itself is mostly textual but can be combined with graphical, tabular, or other notations. A notable property of projectional editors is that they should not be able to produce syntactically invalid input.

A projectional editor can be viewed as halfway in the direction of a form-based editor. While projectional editors still use mostly textual notation and try to mimic the text-editing experience, form-based editors utilize the components of common graphical user interfaces to represent the model using interconnected forms. Users interact with items such as windows, buttons, text fields, and checkboxes while utilizing familiar interactions, e.g., clicking, dragging, and dropping. Such traditional graphical user interfaces can be considered visual editors of the underlying domain-specific language models [5].

There are some recommendations on the use of different language notations, and some of their trade-offs are known (e.g., [6]). Another, different aspect of DSLs is the language implementation approach: implementing a language from scratch or using language workbenches that simplify language definition, either with classical parsing technologies (e.g., Xtext or Spoofax) or with projectional and graphical editing (e.g., JetBrains MPS, or MetaEdit+). A discussion of the advantages and shortcomings of such approaches for language designers already exists [7]. However, we are not aware of any human-based empirical study comparing different editor kinds from the DSL user's point of view.

The main goal and the contribution of the presented work is, therefore, an experimental evaluation of different types of editors for domain-specific languages. We have conducted experiments where we compared three editors for the same DSL:

1. *A text editor* with syntax highlighting and code completion based on Eclipse IDE (Xtext);
2. *A projectional editor* based on JetBrains MPS;
3. *A form-based web editor*.

Our hypothesis for the experiment was that the choice of the editor type would have a significant impact on the language users' productivity.

While the scope of the experiment was quite limited and does not allow us to make definitive conclusions, it provides useful insights into the efficiency of using different kinds of editors for DSLs.

2. Language and Editors

For the purpose of our experiment, we chose a simple DSL for defining quizzes. It is a simple and well-known domain, so it is easier for users to learn the language. In addition, existing form editors are available for the domain.

We chose Google Forms (https://docs.google.com/forms/; accessed on 29 July 2022) as the basis for our language design. We analyzed the tool and extracted a simplified version of the metamodel of the quiz definition language from this editor. This metamodel was used as the basis for developing two other notations of the language and corresponding editors.

The metamodel is shown in Figure 1. A quiz has a name, a description, and a list of questions of different types. Most of the question types have a set of correct and incorrect answers. In addition, each question can have an answer key defining the number of points for correct answers and an optional feedback text.

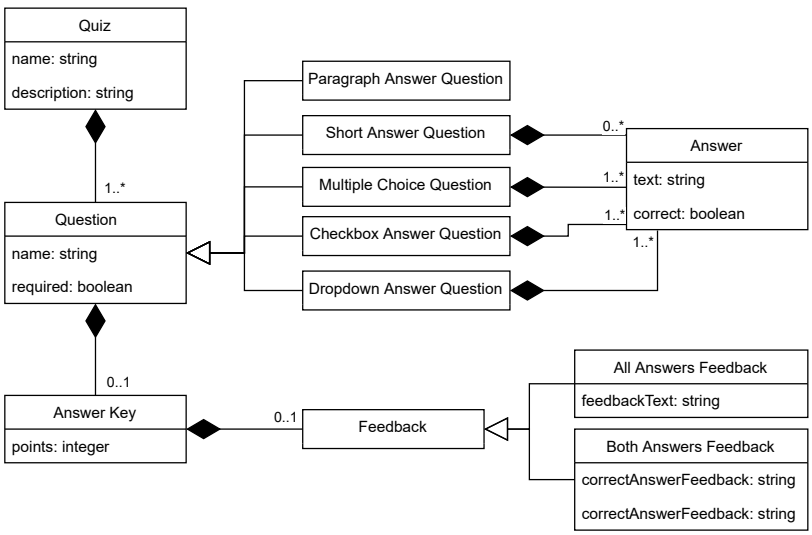

Figure 1. The metamodel of the simplified quiz definition language.

2.1. Form-Based Editor

Google Forms itself is used as an example of a form-based editor for language. Its name, Forms, is purely coincidental in this case, and it is only a specific example of a form-based editor. This web application provides interactive forms for defining quiz properties and questions.

The editor displays questions in the same way as they would be shown to quiz participants. Clicking on the question replaces the preview with the editor form, where the question type, text, and answer choices can be filled. The answer key and feedback are edited in separate forms accessible by buttons. An example quiz definition in the editor is shown in Figure 2.

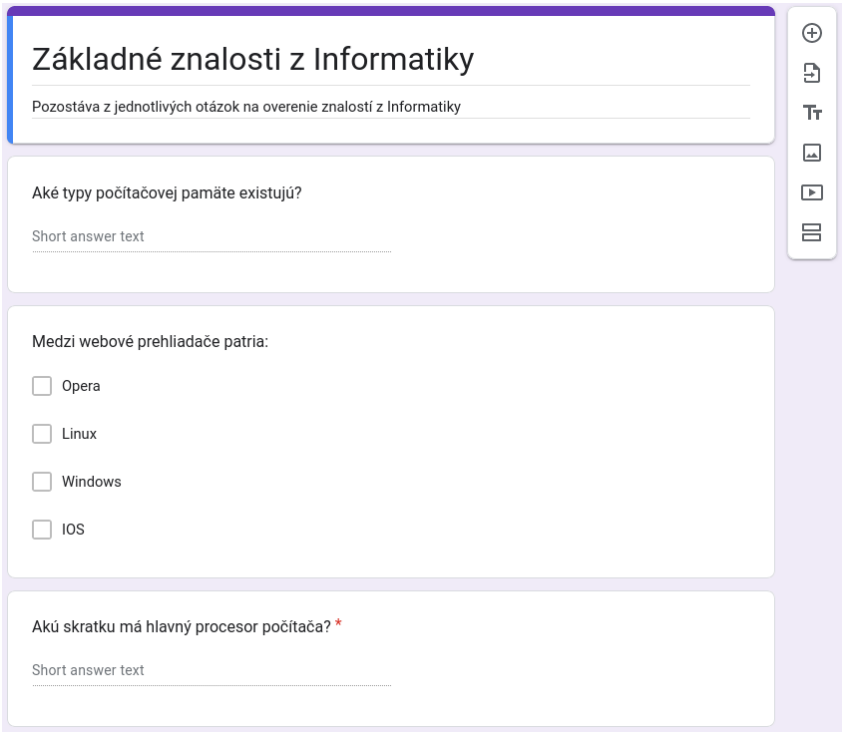

Figure 2. Example quiz in the Google Forms editor.

As Bačíková et al. [5] showed, a GUI (graphical user interface) instance with form items corresponds to a DSL sentence. For example, an input field labeled "Points:" with a value of 5 in a GUI can represent the attribute "points" of an answer key in a model, in the same way as the string "Points: 5" represents it in a text editor. Therefore, we consider a form-based editor a full-fledged alternative to text and projectional editors.

2.2. Advanced Text Editor

Xtext [8] was used as a tool for the implementation of a textual language and a corresponding advanced textual editor. Xtext is a framework for the development of general-purpose and domain-specific languages. It uses language grammar definition to generate a parser, checker, and advanced editor based on the Eclipse IDE or using Language Server Protocol. All of the generated artifacts can be configured or extended using the provided API.

The language model extracted from the Google Forms application (Figure 1) was used to define a textual language using Xtext. Xtext generated a specialized editor for the language with syntax highlighting and code completion (see Figure 3).

```
Test: "Základné znalosti z Informatiky" Description: "Pozostáva z jednoduchých
                                                      otázok na overenie znalostí z informatiky" :
  Short question "Aké typy počítačovej pamäte existujú?" :
      correct answer "ROM"
      correct answer "RAM"
  end
  MultipleChoice question "Medzi webové prehliadače patria:" :
      correct answer "Opera"
      incorrect answer "Linux"
      incorrect answer "Windows"
      incorrect answer "IOS"
          Answer key:
              Points: 5 feedback
               for correct answers: "Opera a Chrome sú jedny z najpopulárnejších prehliadačov."
               for incorrect answers: "Windows a Linux sú operačné systémy."
  end
```

Figure 3. Example quiz in the Xtext-based editor.

2.3. Projectional Editor

As an example of a projectional editor, we chose a leading language workbench that uses this principle—JetBrains MPS (https://www.jetbrains.com/mps/; accessed on 29 July 2022). Language definition in MPS is based on a metamodel. The model becomes the main representation of programs instead of a textual representation. This means that instead of parsing the textual form of a sentence into an internal model, MPS allows editing of the model through the projection.

The metamodel from Figure 1 was replicated in the MPS, and a projectional editor was defined. The editor is similar to the textual editor developed using Xtext. It provides highlighting of keywords and code completion for choosing language elements (see Figure 4). In addition, because the edited form is just a projection of the model, it does not allow entering syntactically invalid input. On the one hand, this guides a user and eliminates a lot of potential errors. On the other hand, it limits a user and requires some specific mode of interaction instead of the flexibility of a plain text editor.

```
Test: Základné znalosti z Informatiky Description: Pozostáva z jednoduchých otázok
                                                   na overenie znalostí z informatiky
  Short question: Aké typy počítačovej pamäte existujú?  Required: false
      ROM correct: true
      RAM correct: true
  <no answerKey>
  MultipleChoice question: Medzi webové prehliadače patra: Required: false
      Opera correct: true
      Linux correct: false
      Windows correct: false
      IOS correct: false
  Answer key:
    Points: 5 feedback for correct answers: Opera a Chrome sú jedny
                                            z populárnych webových prehliadačov.
        feedback for incorrect answers: Linux a Windows sú operačné systémy.
  Short question: Akú skratku má hlavný procesor počítača? Required: true
    << ... >>
  <no answerKey>
```

Figure 4. Example quiz in the MPS-based editor.

3. Method

The study was designed as a controlled experiment complemented by a quantitative and qualitative survey.

3.1. Hypotheses

The controlled experiment was divided into two parts that can be considered separate experiments from the point of view of the statistical analysis. For the first experiment, the hypotheses were as follows:

- $\mathbf{H_{X0}}$: There is no difference in the time spent on the tasks using an advanced text editor (Xtext) compared to a form-based editor (Google Forms).
- $\mathbf{H_{X1}}$: There is a difference in the time spent on the tasks using an advanced text editor (Xtext) compared to a form-based editor (Google Forms).

For the second experiment, the hypotheses were the following:

- $\mathbf{H_{M0}}$: There is no difference in the time spent on the tasks using a projectional editor (MPS) compared to a form-based editor (Google Forms).
- $\mathbf{H_{M1}}$: There is a difference in the time spent on the tasks using a projectional editor (MPS) compared to a form-based editor (Google Forms).

3.2. Variables

In each of the two experiment parts, the independent variable was the type of the editor being used: an advanced text editor, a projectional editor, or a form-based editor.

There was one dependent variable: the time spent on the tasks by the participants, measured in seconds.

3.3. Experiment Design

For each part of the experiment, we used the within-subject design, so each participant performed the specified tasks using two editors:

1. Xtext and Google Forms, or
2. JetBrains MPS and Google Forms.

Because the sequence of the editors in which the experiment participants perform the tasks can influence the results, we used counter-balancing, so the participants were randomly divided into two subgroups with a different order of the editors.

3.4. Tasks

The tasks were designed in such a way that they would be easy to complete for all participants. This allowed us to measure and compare only the time of completion.

Participants were given an existing quiz with 3 questions. Then they needed to perform 3 tasks with subtasks:

1. Small changes in existing questions and answers: make a specific question required, change the text of one of the answers and mark it as correct, and change the number of points for a specific question.
2. Addition of elements into an existing question: a new answer, the number of points for the question, and feedback for a correct answer were added.
3. Addition of new question: the question type, text, and answers were specified.

The tasks allowed participants to gradually become familiar with the language and the editor and also covered different kinds of typical editing needs—both reading and modifying the existing model, and adding new elements.

3.5. Survey

The survey contained four questions about each of the editors. The participants needed to assess on the 5-point Likert scale how simple or difficult it was to

1. Change the parameters of an existing question;
2. Add new parameters to an existing question;
3. Define a completely new question;
4. Learn how to work with the editor.

The scale was from 1—simple to 5—difficult.

After these questions, there was a free-form question about their opinion on the editors and which of them they would choose in the future.

3.6. Participants

The participants of the study were graduate students of Computer Science at the Technical University of Košice studying the Metaprogramming course. There were 39 participants in total. Two of them were excluded from the results: one performed the tasks in both editors at the same time, and the other did not complete the tasks. As a result, we had 37 participants who completely finished the tasks: 20 in the Xtext vs. Forms group, and 17 in the MPS vs. Forms group.

3.7. Setup

Because of the restrictions connected to the COVID-19 pandemic, the experiment was performed online.

The subjects connected to a Windows Server 2019 virtual machine. User accounts were created on this computer for all participants in the experiments. The following programs and files had been prepared for each user account and were launched after logging in:

- OBS Studio for screen recording;
- WordPad with the specification of the experiment tasks;
- Jetbrains MPS or Eclipse Xtext with a prepared simple quiz for performing tasks;
- A web browser with instructions for the tested domain-specific language and editors, a Google Forms quiz for performing tasks, and an experiment survey.

3.8. Procedure

The instructions for performing the experiment were sent to the participants via email, which included the instructions for connecting to the virtual machine together with the login details for the user account.

After connecting to the virtual machine using a remote desktop, the participants were able to study the prepared documentation and tasks. The documentation included a short tutorial about the creation and editing of questions in both editors assigned to the given participant.

When the subjects were prepared to begin their tasks, they needed to turn on screen recording to evaluate the time required to complete the tasks. They performed the specified task in two editors in the given order. After completion, they turned off the recording and filled out the final survey.

3.9. Data Collection

Two forms of data were collected: screen recordings of the participants performing the tasks and the results of the final survey. For each user account, we evaluated the screen recording and measured the time of completion of the tasks in the individually tested editors.

3.10. Statistical Analysis

For each experiment part and each value of the independent variable, we displayed the values of the dependent variable on a histogram and a normal Q–Q plot. Since the data were not normally distributed, we used the Wilcoxon signed-rank test to assess statistical significance. We chose a standard value of α (0.05).

4. Results

In this section, we first present the results of both parts of the controlled experiment. Then, we summarize the quantitative and qualitative findings from the survey.

4.1. Controlled Experiment

For the experiment comparing an advanced text editor and a form-based editor, the collected results are displayed textually in Table 1 and graphically in Figure 5.

The median time spent on the tasks in the Xtext group was 381 s, while in the Google Forms group it was 390.5 s (2.43% faster). The Google Forms group had a longer upper tail, so the difference in means is larger: 382.95 s for an advanced text editor and 432.40 s for a form-based editor (11.44%).

The computed p-value was 0.1054, which is more than α (0.05). Therefore, the difference between the groups was not statistically significant.

The results of the experiment comparing a projectional and form-based editor can be seen in Table 2 and Figure 6.

Table 1. Results of the XText vs. Google Forms experiment.

ID	Xtext (s)	Google Forms (s)
1	416	450
2	345	255
3	310	300
4	288	327
5	356	380
6	466	549
7	488	490
8	447	354
9	418	339
10	406	336
11	480	561
12	414	531
13	610	748
14	292	386
15	206	255
16	260	355
17	321	395
18	204	540
19	595	430
20	337	667
Median	381.00	390.50
Mean	382.95	432.40
Std. Dev.	112.62	133.33

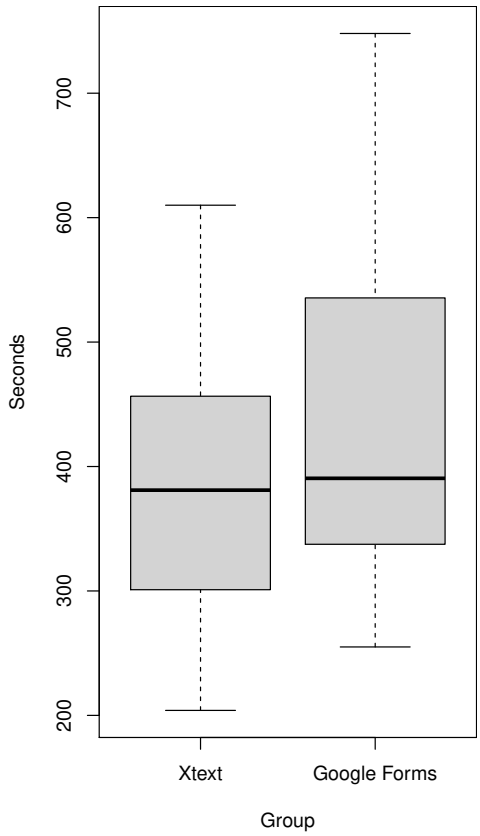

Figure 5. Results of the XText vs. Google Forms experiment.

Table 2. Results of the MPS vs. Google Forms experiment.

ID	MPS (s)	Google Forms (s)
1	243	217
2	276	277
3	323	387
4	260	462
5	428	276
6	387	378
7	393	326
8	559	374
9	303	381
10	726	576
11	841	461
12	395	411
13	365	330
14	707	389
15	613	674
16	572	370
17	270	263
Median	393.00	378.00
Mean	450.65	385.41
Std. Dev.	184.69	113.46

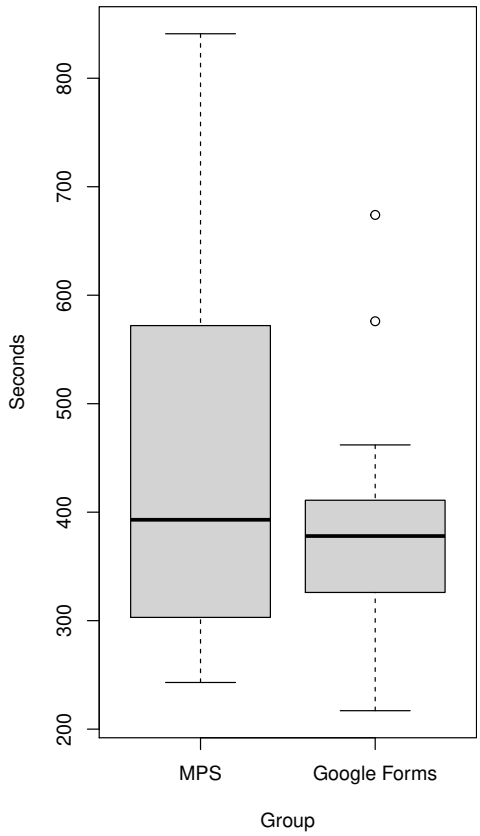

Figure 6. Results of the MPS vs. Google Forms experiment.

In the MPS group, the median time to complete the tasks was 393 s. The Google Forms group achieved a median of 378 s, which is 3.82% faster. We observed a much larger variance in the MPS group, especially in the upper tail. Therefore, the means are 450.65 s for MPS compared to 385.41 s in the Google Forms group. The relative difference of means was 14.48%.

In this second experiment, we computed a p-value of 0.1350, which means the difference between groups was not significant.

After combining the results, the best times were measured for the Xtext editor followed by Google Forms and finally MPS. However, the differences were not statistically significant, so we cannot reject the null hypotheses ($\mathbf{H_{X0}}$, $\mathbf{H_{M0}}$) with enough confidence.

This does not mean, however, that the three editor approaches do not differ in editing efficiency in general. Negative or inconclusive results relate only to the specific context that was tested [9]. In our case, the most probable cause of the obtained results was the simplicity of the language on one side and the proficiency of the subjects on the other side. The participants were master's degree computer science students, often with professional programming experience, while DSLs are usually targeted at non-programming domain experts. Subjects without any previous programming experience often fail to cope with concepts that are obvious to developers, such as nested parentheses [10]. Therefore, we could expect different results for other participant populations.

4.2. Quantitative Survey Results

Subjective evaluation of the editors provided by the participants in the survey is summarized in Figures 7 and 8. Diagrams show the number of each answer type on the Likert scale, where 1 means simple and 5 means difficult.

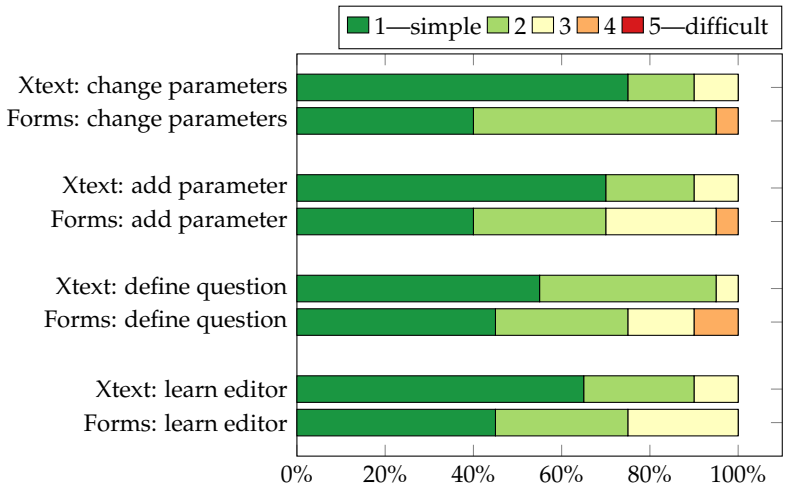

Figure 7. Survey results of the Xtext vs.Google Forms experiment.

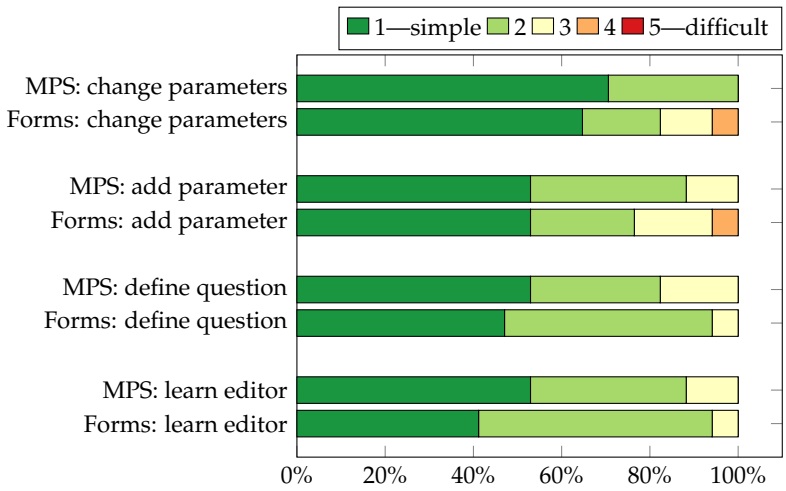

Figure 8. Survey results of the MPS vs.Google Forms experiment.

Participants considered all aspects of editing to be rather simple. Xtext and MPS had slightly more answers on the simple side of the scale, while Google Forms was quite difficult for several participants.

Participants also selected which editor they would choose in the future to define a quiz. Their choices are shown in Figure 9. In the first experiment, the participants preferred Xtext (14 votes) over Google Forms (6 votes). In the second one, they preferred MPS (11 votes) over Google Forms (6 votes).

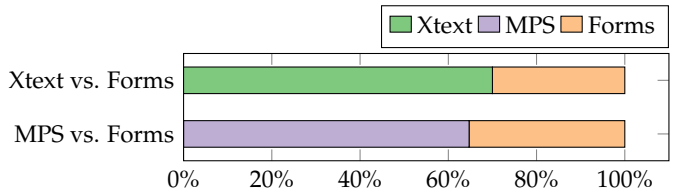

Figure 9. Editor type students would prefer in the future.

4.3. Qualitative Survey Results

In addition to their preference, we asked participants about the reasons why they would prefer a specific editor in the future.

4.3.1. Text Editor

Xtext was quite popular in the participants' answers. The main reason they mentioned was the efficiency of working with a keyboard instead of a mouse. In addition, the textual representation allowed for simple copying of parts of the quiz definition.

They also mentioned better clarity of the textual representation: all details of questions are directly visible and do not require switching between separate forms as in the case of Google Forms.

Plain-text editors allow the users to easily customize the visual appearance of a DSL sentence being constructed by using spaces, tabulators, and newlines. Thanks to this, certain constructs can be aligned horizontally to resemble table-like structures. Of course, this only holds for whitespace-insensitive languages.

A particularly interesting advantage was mentioned by one subject: the plain-text format is more straightforward to synchronize and merge across multiple versions. This makes it the preferred format when the collaboration of multiple persons is expected. However, ongoing research in the area of conflict-free replicated data types [11] aims to enable the synchronization of more complicated data structures without conflicts.

Another advantage of textual languages is the possibility to copy and paste arbitrary portions of code without any limitations. In contrast to this, projectional editors have specific rules on which portions can be copied and how, and in form-based editors, the user depends entirely on the options provided by the GUI creator.

Finally, users' confidence in their actions and the effects they produce also plays a role in the selection of the preferred editor type. As one participant stated about Xtext: "I have the feeling of greater control over what I do".

4.3.2. Projectional Editor

Similar to Xtext, the participants positively assessed the effectiveness of keyboard control and the clarity of the textual notation. Some of them mentioned that the editing was intuitive.

Other participants, however, considered the projectional editing style to be unintuitive. They explained that a user must be more careful with the editor because it is easy to delete some elements by mistake.

The absence of shortcuts from traditional text editors, such as Ctrl+Backspace to delete the last word, tended to be criticized. Some subjects also suggested that specific actions, e.g., navigation to the nearest unfilled parameter, should have domain-specific keyboard shortcuts, which could improve user experience.

The projectional editor was also praised for standardized error reporting. All errors are marked with a consistent visual style. It is also possible to list all issues, regardless of their type, in one simple view. On the other hand, in form-based editors, the error reporting style depends completely on the choice of the DSL developer.

Preferring a given editor type can be also a matter of culture. One subject responded, simply, "MPS, I'm a geek".

4.3.3. Form-Based Editor

The main advantage of Google Forms mentioned by our participants was the fact that users do not need to learn and remember the syntax and keywords of the language. All available options are presented in the forms that users can fill or select from the available options. Therefore, the interface is easy to learn and intuitive.

Another important advantage is that users can directly see the results of their work—the defined quiz. Although a preview of the result can be integrated into other types of editors as well, this is partially an inherent property of some form-based editors in specific domains. Particularly in our domain, a graphical user interface used to produce the quiz can seamlessly resemble the GUI that the quiz-taker will utilize, which had an advantage in Google Forms.

One subject compared the selection between a form-based and a text (or projectional) editor to the selection between Microsoft Word and LaTeX for thesis writing. Form-based GUIs are easier to learn for non-programmers, while textual interfaces manifest their advantages over long-time usage when the user becomes more comfortable with the notation and starts using advanced language features.

As a disadvantage of Google Forms, some users mentioned the need to "click through" the interface. The need to use a mouse for selecting options and switching forms is less efficient for larger quizzes.

5. Threats to Validity

Now, we describe the threats to the validity of the performed study, using the guidelines by Wohlin et al. [12].

5.1. Construct Validity

The study consisted of two separate experiments using within-subject design. Instead of this, we could perform a single experiment with between-subject design. This would increase the number of participants assigned to the Xtext and MPS groups, thus increasing the probability of obtaining statistically significant results. On the other hand, by offering each participant two different editors, we increased the validity of the survey, since the participants answered the questions immediately after the completion of the same tasks in both editors.

In the quantitative survey rating the simplicity of the individual aspects of each editor, we offered the participants a five-point Likert scale. A vast majority of answers was 1 (very simple) or 2 (simple), while none of the participants selected 5 (very difficult) in any of the questions. Offering a more fine-grained Likert scale or asking more specific questions could improve the diversity of the results. On the other hand, even from the current survey, we can see that the form-based editor received a slightly worse rating for some aspects.

5.2. Internal Validity

Since we used a within-subject design, and each participant performed the same tasks on the same language with two different editors, the learning effect was certainly present. However, we mitigated its negative influence on validity by using counter-balancing: we alternated the order of the editors used by the subjects.

Due to the COVID-19 pandemic, the experiment had to be performed remotely. This could affect the performance of the participants. However, we supplied them with all the necessary materials to perform the tasks successfully. Screen recording could cause

the Hawthorne effect, i.e., change the behavior of the participants because of observation. Nevertheless, it allowed us to ensure the execution was performed according to the protocol and was not randomly interrupted.

5.3. External Validity

All participants of the study were computer science students enrolled in an elective course focused on metaprogramming. This means they had prior programming experience and therefore might prefer advanced editors offering richer keyboard interaction (text or projectional editors) to simple graphical user interfaces (form-based editors). In the future, it would be interesting to perform a similar study with subjects without any prior programming experience and compare the results.

The tasks used in the study were relatively simple. This could be perceived as an advantage since it allowed the subjects to complete them without difficulties, and we could measure solely completion times instead of considering also success rates. On the other hand, more difficult tasks could make the differences between the individual editors more prominent.

5.4. Reliability

By performing the experiment online using pre-supplied materials, we practically eliminated the influence of the researchers on participants.

All materials, including the tutorials, task definitions, language source code, raw results, and processing scripts are permanently available to the public at https://doi.org/10.17605/OSF.IO/RWSP6 (accessed on 12 August 2022).

5.5. Conclusion Validity

We did not observe large differences between the groups in the controlled experiment, and the results were not statistically significant. However, the publication of negative results is considered a very important aspect to avoid bias [13].

Furthermore, we complemented the experiment with a survey that provided both quantitative results and valuable qualitative explanations as to why each of the editors was considered better or worse by the participants.

6. Related Work

The design and development of a domain-specific language is a well-researched topic [14]. Most of the research, however, has focused on the textual or graphical notations of DSLs. With the introduction of language workbenches with projectional editors, such as Intentional software [15] and JetBrains MPS, projectional editors have become a viable alternative.

Projectional editors have been used in a wide range of languages, including a complex general-purpose language—extensions of the C language for embedded software development [16]. They have also been used as an alternative to form-based user interfaces. For example, JetBrains MPS was used to develop an interface for biological data analysis [17] or software requirements modeling [18].

The use of form-based editors as an alternative to a textual DSL has been demonstrated, for example, for a security policy specification language [19]. IIS*Case tool [20,21] is using a combination of form-based, graphical, and textual languages for model-driven development of information systems. Some DSL development tools also support form-based editors, including Melanee [22] and EMF Forms [23].

Despite the collected experience, empirical evaluation and comparison of different types of editors are still limited. This is related to the overall lack of empirical evaluation in DSL research [24].

There are several studies comparing domain-specific to general-purpose languages. For example, Kosar et al. [25] conducted an experiment where they compared domain-specific and general-purpose languages for the task of graphical user interface definition.

Participants of the experiments were given tests examining their ability to learn the used notation and understand the meaning of a program and modify it. The success rate of program comprehension was better in the case of DSL. Deeper analysis based on a cognitive dimensions framework showed that DSL was superior in all cognitive dimensions.

In the follow-up study, Kosar et al. [26] used three different domains: feature modeling, graph description, and graphical user interfaces. They again found that the correctness of answers to test questions was significantly higher in the case of DSL compared to a general-purpose language. Participants also needed less time to complete the tests; thus, the efficiency of program comprehension was also significantly higher in the case of DSL.

They, however, intentionally excluded the influence of editors on the results in both of the experiments. For this reason, the tests were taken on paper, so no software support tools or specific editors were used. All languages used in the experiment had textual notation.

Barišić et al. [27] also compared DSL to a general-purpose language. It was a graphical DSL for the domain of High Energy Physics on the one hand, and C++ on the other hand. The effectiveness, efficiency, and confidence of the users were significantly better in the case of the DSL, but it was not clear how much the results were influenced by the editor of the language.

Another similar study was conducted by Johanson and Hasselbring [28]. They used the domain of marine ecosystem simulation, and the participants were domain experts—ecologists. The compared languages were Sprat Ecosystem DSL and C++. They found significant improvement in correctness and time needed to complete the tasks using the DSL. As an editor, however, they used a simple, web-based text editing component with syntax highlighting for both languages.

Nosáľ et al. [10] investigated the influence of IDE features on the use of embedded DSLs. They compared a DSL based on the Ruby language, known for flexible syntax with a minimum of syntactic noise, with a DSL based on Java with more strict syntax. The Java-based language, however, used several IDE features to improve the editing experience, including generating the boilerplate code, hiding fragments of the code with syntactic noise and preventing their editing, code completion, and custom error reporting. Both languages were tested on non-programmers. As a result, the users with the Java-based language with the customized IDE achieved significantly better correctness than the users with the Ruby-based language and a standard IDE. This shows that the editor features were able to overcome the advantages of the flexible syntax of Ruby.

The difference between projectional and parser-based editors was measured in an experiment by Berger et al. [29]. It compared the editing of C programs in the projectional editor based on JetBrains MPS and in the parser-based editor Eclipse CDT. The results showed that for basic editing, a projectional editor provides comparable efficiency to the parser-based editor. In more advanced tasks, such as refactoring, the projectional editor is more limiting. They focused on a general-purpose programming language instead of a DSL. For this reason, it did not make sense to compare it with a form-based editor.

Lopes et al. [30] performed an experiment comparing graphical and textual DSLs for conceptual database modeling. In their case, they did not find a significant difference in the time needed to perform tasks and effectiveness.

There are also studies focused on a specific feature of code editors. For example, Hannebauer et al. [31] evaluated the influence of syntax highlighting on program comprehension by novice programmers. They found no evidence that syntax highlighting improves the correctness of code comprehension tasks.

7. Conclusions and Future Work

We presented an empirical study comparing three different editors of the same domain-specific language: an advanced plain-text editor implemented in the Xtext framework, a projectional editor developed using JetBrains MPS, and a form-based GUI editor, namely Google Forms. The topic of the DSL was the definition of quizzes.

The study consisted of a controlled experiment with human subjects divided into two groups (Xtext vs. Google Forms and MPS vs. Google Forms) and an accompanying survey. The task of the participants was to perform small changes to existing questions, add elements to an existing question, and add a new question. We measured the time spent on the tasks.

In the controlled experiment, the measured time for Xtext was marginally lower than for Google Forms (2.4% difference in median times, 11.4% difference in mean times), and Google Forms had a slightly lower measured time than MPS (3.8% medians, 14.5% means). However, the results were statistically insignificant, so we cannot conclude which editor type achieved the best efficiency. We attribute the smallness of the differences mainly to the programming expertise of the participants combined with a simple DSL.

In the quantitative survey, questions focused on the ease of specific aspects of editing; the participants rated all editors rather favorably. A small exception was Google Forms, which some subjects considered not so easy for question parameter addition. When asked about which editor they would prefer in the future, the majority of subjects selected Xtext or MPS, with Google Forms being less popular.

The higher preference for MPS over Google Forms seems to contradict the efficiency results. Such disparity between objective and subjective evaluation has already been observed in other works [32]. In our case, the survey question was about their future preference, so the respondents might have been trying to predict their efficiency in long-term usage. This assumption is also supported by some of their answers in the qualitative survey. On the other hand, it may have just been their personal preferences.

We also obtained interesting qualitative data. Xtext and MPS were praised for their efficient, mouse-free interaction. In Xtext, the possibility to copy, paste, and synchronize plain-text data easily during collaboration, and the "feeling of control" were highlighted. MPS created a controversy, with a portion of the subjects being satisfied with its keyboard controls and other participants demanding the support of all traditional keyboard shortcuts known from plain-text editors. Special domain-specific shortcuts were also suggested. Google Forms was considered easy to learn and use for non-programmers. It relieved the users of the necessity to learn and remember syntax elements, but many participants would prefer a text-based editor in a longer time frame.

It is important to note that we consider this study only the first in a potential series of experiments. Multiple differentiated replications should be performed in the future. Possible variations include slightly modified language definitions or completely diverse domains, more difficult tasks, and measurement of other variables. Different variations and implementations of individual editor types might be tested, too, e.g., a text editor without advanced syntax highlighting and code completion. Using domain experts without any prior programming experience would be particularly interesting, since we hypothesize that their preferences differ from those of programmers to a great extent. Furthermore, this would mitigate the possible influence of prior Eclipse or MPS experience on the results. Another potential area of investigation is the long-term usage of the editors, because users' efficiency with some types of editors might change with growing experience.

Author Contributions: Conceptualization, J.P.; methodology, S.C., M.S., M.K. and J.P.; software, S.C. and M.K.; validation, M.S. and M.K.; formal analysis, M.S. and M.K.; investigation, M.K.; resources, S.C.; data curation, S.C., M.S. and M.K.; writing—original draft preparation, S.C., M.S. and M.K.; writing—review and editing, S.C., M.S. and J.P.; visualization, S.C. and M.S.; supervision, S.C.; funding acquisition, J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by VEGA grant no. 1/0630/22, Lowering Programmers' Cognitive Load Using Context-Dependent Dialogs.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The data presented in this study are openly available at https://doi.org/10.17605/OSF.IO/RWSP6 (accessed on 12 August 2022).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Voelter, M. Best practices for DSLs and model-driven development. *J. Object Technol.* **2009**, *8*, 79–102.
2. Sulír, M.; Bačíková, M.; Chodarev, S.; Porubän, J. Visual augmentation of source code editors: A systematic mapping study. *J. Vis. Lang. Comput.* **2018**, *49*, 46–59. [CrossRef]
3. Sulír, M.; Porubän, J. Augmenting Source Code Lines with Sample Variable Values. In Proceedings of the 26th Conference on Program Comprehension, Association for Computing Machinery, ICPC'18, New York, NY, USA, 28–29 May 2018; pp. 344–347. [CrossRef]
4. Voelter, M.; Siegmund, J.; Berger, T.; Kolb, B. Towards User-Friendly Projectional Editors. In *Software Language Engineering*; Lecture Notes in Computer Science; Combemale, B., Pearce, D., Barais, O., Vinju, J., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; Volume 8706, pp. 41–61. [CrossRef]
5. Bačíková, M.; Porubän, J.; Lakatoš, D. Defining Domain Language of Graphical User Interfaces. In Proceedings of the 2nd Symposium on Languages, Applications and Technologies, Porto, Portugal, 20–21 June 2013; OpenAccess Series in Informatics (OASIcs); Leal, J.P., Rocha, R., Simões, A., Eds.; Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik: Dagstuhl, Germany, 2013; Volume 29, pp. 187–202. [CrossRef]
6. Zdun, U.; Strembeck, M. Reusable Architectural Decisions for DSL Design. In Proceedings of the 14th European Conference on Pattern Languages of Programs (EuroPLoP), Irsee, Germany, 8–12 July 2009; pp. 1–37.
7. Erdweg, S.; Storm, T.; Völter, M.; Boersma, M.; Bosman, R.; Cook, W.; Gerritsen, A.; Hulshout, A.; Kelly, S.; Loh, A.; et al. The State of the Art in Language Workbenches. In *Software Language Engineering*; Lecture Notes in Computer Science; Springer International Publishing: Berlin/Heidelberg, Germany, 2013; Volume 8225, pp. 197–217. [CrossRef]
8. Eysholdt, M.; Behrens, H. Xtext - Implement your Language Faster than the Quick and Dirty way. In Proceedings of the ACM International Conference Companion on Object-oriented Programming, Systems, Languages, and Applications—SPLASH'10, Reno, NV, USA, 17–21 October 2010; ACM Press: New York, NY, USA, 2010; p. 307. [CrossRef]
9. Guéhéneuc, Y.G.; Khomh, F. Empirical Software Engineering. In *Handbook of Software Engineering*; Springer International Publishing: Cham, Switzerland, 2019; pp. 285–320. [CrossRef]
10. Nosáľ, M.; Porubän, J.; Sulír, M. Customizing Host IDE for Non-programming Users of Pure Embedded DSLs: A Case Study. *Comput. Lang. Syst. Struct.* **2017**, *49*, 101–118. [CrossRef]
11. Kleppmann, M.; Wiggins, A.; van Hardenberg, P.; McGranaghan, M. Local-First Software: You Own Your Data, in Spite of the Cloud. In Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2019, Athens, Greece, 23–24 October 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 154–178. [CrossRef]
12. Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslén, A. *Experimentation in Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2012.
13. Romano, S.; Fucci, D.; Scanniello, G.; Teresa Baldassarre, M.; Turhan, B.; Juristo, N. Researcher Bias in Software Engineering Experiments: A Qualitative Investigation. In Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portorož, Slovenia, 26–28 August 2020; pp. 276–283. [CrossRef]
14. Mernik, M.; Heering, J.; Sloane, A.M. When and how to develop domain-specific languages. *ACM Comput. Surv.* **2005**, *37*, 316–344. [CrossRef]
15. Simonyi, C.; Christerson, M.; Clifford, S. Intentional software. In Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications, Portland, OR, USA, 22–26 October 2006; pp. 451–464.
16. Voelter, M.; Kolb, B.; Szabó, T.; Ratiu, D.; van Deursen, A. Lessons learned from developing mbeddr: A case study in language engineering with MPS. *Softw. Syst. Model.* **2019**, *18*, 585–630. [CrossRef]
17. Benson, V.M.; Campagne, F. Language workbench user interfaces for data analysis. *PeerJ* **2015**, *3*, e800. [CrossRef] [PubMed]
18. Savić, D.; Vlajić, S.; Lazarević, S.; Antović, I.; Stanojević, V.; Milić, M.; Silva, A. SilabMDD: A use case model driven approach. In Proceedings of the ICIST 2015 5th International Conference on Information Society and Technology, Kopaonik, Serbia, 8–10 March 2015.
19. Harnoš, P.; Dedera, L. Analysis of current trends in the development of DSLs and the possibility of using them in the field of information security. *Sci. Mil. J.* **2021**, *16*, 15–27. [CrossRef]
20. Luković, I.; Mogin, P.; Pavićević, J.; Ristić, S. An approach to developing complex database schemas using form types. *Softw. Pract. Exp.* **2007**, *37*, 1621–1656. [CrossRef]
21. Luković, I.; Pereira, V.J.M.; Oliveira, N.; Henriques, R.P. A DSL for PIM specifications: Design and attribute grammar based implementation. *Comput. Sci. Inf. Syst.* **2011**, *8*, 379–403. [CrossRef]
22. Atkinson, C.; Gerbig, R. Flexible deep modeling with Melanee. In Proceedings of the Modellierung 2016 Workshopband, Karlsruhe, Germany, 2–4 March 2016; Gesellschaft für Informatik eV: Bonn, Germany, 2016; pp. 117–121.
23. Jafer, S.; Chhaya, B.; Durak, U. Graphical specification of flight scenarios with aviation scenario defintion language (ASDL). In Proceedings of the AIAA Modeling and Simulation Technologies Conference, Grapevine, TX, USA, 9–13 January 2017; p. 1311.

24. Kosar, T.; Bohra, S.; Mernik, M. Domain-Specific Languages: A Systematic Mapping Study. *Inf. Softw. Technol.* **2016**, *71*, 77–91. [CrossRef]
25. Kosar, T.; Oliveira, N.; Mernik, M.; Pereira, V.J.M.; Črepinšek, M.; Da Cruz, D.; Henriques, R.P. Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. *Comput. Sci. Inf. Syst. ComSIS* **2010**, *7*, 247–264. [CrossRef]
26. Kosar, T.; Mernik, M.; Carver, J.C. Program comprehension of domain-specific and general-purpose languages: Comparison using a family of experiments. *Empir. Softw. Eng.* **2012**, *17*, 276–304. [CrossRef]
27. Barišić, A.; Amaral, V.; Goulão, M.; Barroca, B. Quality in Use of Domain-Specific Languages: A Case Study. In Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools—PLATEAU'11, Portland, OR, USA, 24 October 2011; ACM: New York, NY, USA, 2011; p. 65. [CrossRef]
28. Johanson, A.N.; Hasselbring, W. Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: A controlled experiment. *Empir. Softw. Eng.* **2017**, *22*, 2206–2236. [CrossRef]
29. Berger, T.; Völter, M.; Jensen, H.P.; Dangprasert, T.; Siegmund, J. Efficiency of projectional editing: A controlled experiment. In Proceedings of the International Symposium on Foundations of Software Engineering—FSE 2016, Seattle, WA, USA, 13–18 November 2016; ACM Press: New York, NY, USA, 2016; pp. 763–774. [CrossRef]
30. Lopes, J.; Bernardino, M.; Basso, F.; Rodrigues, E. Textual-based DSL for Conceptual Database Modeling: A Controlled Experiment. In Proceedings of the Anais do XXXVI Simpósio Brasileiro de Banco de Dados (SBBD 2021), Rio de Janeiro, Brazil, 4–8 October 2021; Sociedade Brasileira de Computação—SBC: Porto Alegre, Brazil, 2021; pp. 169–180. [CrossRef]
31. Hannebauer, C.; Hesenius, M.; Gruhn, V. Does syntax highlighting help programming novices? *Empir. Softw. Eng.* **2018**, *23*, 2795–2828. [CrossRef]
32. Fabry, J. The Meager Validation of Live Programming. In Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming, Association for Computing Machinery, Programming'19, Genova, Italy, 1–4 April 2019. [CrossRef]