Evaluation of an Integrated Tool Environment for Experimentation in DSL Engineering

Florian Häser, Michael Felderer(✉), and Ruth Breu

Department of Computer Science, University of Innsbruck, Innsbruck, Austria
{florian.haeser,michael.felderer,ruth.breu}@uibk.ac.at

Abstract. Domain specific languages (DSL) are a popular means for providing customized solutions to a certain problem domain. So far, however, language workbenches lack sufficient built-in features in providing decision support when it comes to language design and improvement. Controlled experiments can provide data-driven decision support for both, researchers and language engineers, for comparing different languages or language features. This paper provides an evaluation of an integrated end-to-end tool environment for performing controlled experiments in DSL engineering. The experimentation environment is presented by a running example from engineering domain specific languages for acceptance testing. The tool is built on and integrated into the Meta Programming System (MPS) language workbench. For each step of an experiment the language engineer is supported by suitable DSLs and tools all within the MPS platform. The evaluation, from the viewpoint of the experiments subject, is based on the technology acceptance model (TAM). Results reveal that the subjects found the DSL experimentation environment intuitive and easy to use.

Keywords: Domain specific languages · DSL · Language engineering
Experimentation · Model-based software engineering
Meta Programming System · Empirical software engineering

1 Introduction

Domain specific languages (DSLs) are powerful languages used to build customized solutions for certain problem domains. In model-driven software engineering they are used because they provide less complexity and enable easier communication with domain experts [1]. They are not only widely used in practice [2], but also well researched [3,4].

A recent comprehensive systematic mapping study on DSL engineering conducted by Kosar et al. [4] revealed that within the DSL community, researchers only rarely evaluate their languages. Although all steps of the language engineering process are important, researchers mostly focus on the design and implementation, rather than domain analysis and maintenance. The mapping study provides evidence which shows that DSLs are rarely validated by taking end-user feedback into account. Thus researchers assume that they developed a DSLs perfectly tailored for the domain. Similar results are provided by Carver et al. [5] and Gabriel et al. [6].

The Meta Programming System (MPS) [7], an industry-ready language workbench, supports language engineers in the design and implementation phase. This includes the definition of syntax, semantics and the creation of language specific editors. Those editors bring remarkable benefit for the language user in defining DSL constructs.

So far, language workbenches do not provide sufficient decision support when it comes to domain analysis and language adaptation during maintenance. This is, however, essential for successful language engineering. A language engineer does not always know in advance, what effects a certain adaptation will have to a DSL. For instance, he or she can not know in advance before doing some experimentation if a certain adaptation makes a language really better with respect to aspects such as creation time or resulting document length of the scripts written in the adapted DSL.

Controlled experiments [8] allow to compare different DSL features and provide data-driven decision support for both language engineers and researchers. In the last decade, the number of conducted experiments in the field of software engineering increased. This significantly increased the body of knowledge on limitations and benefits of software engineering methods and techniques [8].

As mentioned above, this is, however, not entirely true for the field of DSL engineering, where experiments are still rare [4].

We assume that one major reason for this is the missing integrated tool support which enables especially language engineers in practice but also researchers to conduct controlled experiments directly within a DSL workbench in a straightforward and methodologically sound way. Such an integrated end-to-end tool environment also enables easy replication of experiments. This is the case because with a (semi-)formal definition the experimental setting can automatically be interchanged and reproduced.

Freire et al. [9] conducted a systematic literature review on automated support for controlled experiments. They discovered that there are already methodologies and tools available, which support parts or even the entire experimentation process. Those approaches, however, are not suitable for experiments with DSLs. They often only provide a DSL to formulate experiments, which is a useful input for us. DSL engineering is not performed on the web and its empirical analysis includes not only questionnaires but also automatic measurements, as only produced and collected in a real language workbench.

This paper contributes the empirical evaluation of an integrated tool environment for performing controlled experiments in DSL engineering. The integrated tool environment itself has been presented already as tool paper at EASE'16 [10], where it received a best tool paper award, and is summarized in this paper to provide the context for the presented empirical study. We are not proposing yet another DSL suited for experimentation, instead we have integrated existing DSLs and extended the DSL engineering environment MPS to cover language experimentation and to support the maintenance phase of the language engineering process.

MPS supports a mixed notation which includes textual, tabular and graphical representation of concepts. Our environment supports the language engineer with DSL design (the experimental object in our case) and all other phases of controlled experiments. This includes planning, operation, analysis & interpretation, and presentation & package. We based our platform on already established DSLs and tools which have been all integrated into MPS.

For the planning of experiments we base our work on the established DSL ExpDSL [11]. The language is used to formulate goals and hypotheses. For statistical methods metaR [12] is reused to simplify the application of complex R-functions for non-experts like most language engineers. The experiment operation is backed up by a MPS extension able to monitor and collect metrics like creation time or document length. For performing the analysis & interpretation phase, R output is displayed in the MPS editor and raw data can be exported as comma separated values (CSV). Finally, for presentation & package a DSL for LaTeX is applied [13]. The integration of several DSLs and representation is enabled by the projectional editors of MPS. They allow the mixture of languages and alternative representations of data, such as plots. Everything together can be displayed within one single document (see Fig. 4). The full integration enables researchers and practitioners to conduct controlled experiments in an easy way. All without media breaks and with different levels of rigor.

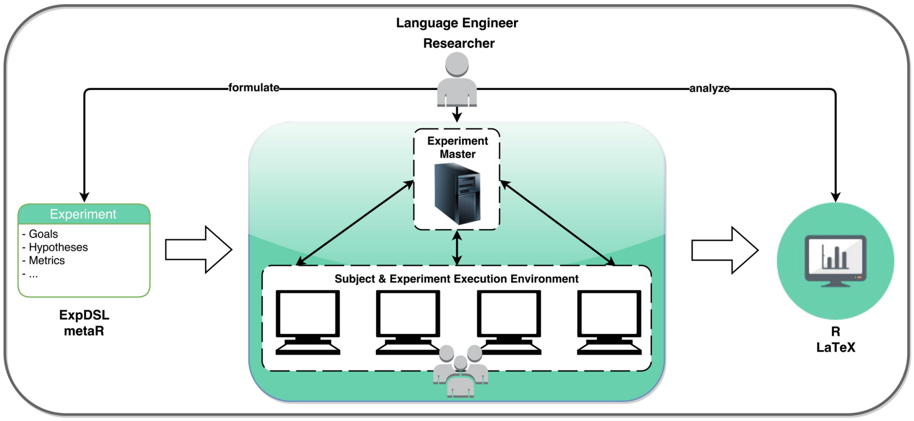


Fig. 1. Integrated DSL experiment environment architecture [10]

The remainder of this paper is organized as follows. Section 2 gives a rough overview on related approaches. Section 3 presents an example experiment to demonstrate DSL experimentation in the domain of acceptance testing of web applications. The example is used as a running example in order to explain the environment. Section 4 provides an overview of the architecture, the underlying workflow and the tool implementation of the integrated experimentation environment for DSL engineering. Sections 3 and 4 are based on a previous tool paper [10] presented at EASE'16. For better understandability the case from Sect. 3 is applied to the steps on conducting a controlled experiment. Section 5 presents a qualitative evaluation of the experiment platform. Finally, Sect. 6 concludes the paper and presents future work and visions for the tool environment.

2 Related Work

In this section we present related work with regard to evaluation of DSLs and experimentation environments for software engineering artifacts.

As already mentioned in the previous section Kosar et al. [4] did a comprehensive literature review on the DSL engineering research field. They empirically discovered that researchers are not evaluating their languages. There may be various reasons for that issue, among them missing tool support. Some recent publications provide frameworks and tools for evaluating languages before and after launching them. Bariŝić et al. [14] have elaborated a usability evaluation framework for DSLs. In the framework they propose the use of controlled experiments for evaluating languages. Izquierdo et al. [15] also propose that the involvement of the real end-user is essential for building and evaluating DSLs.

Tairas and Cabot [16] on the contrary presented a post-hoc evaluation analysis methodology. The DSL is analyzed after launching it based on its actual usage. It gives the language engineer insights on language artifacts that need improvement. Even though they have evaluated their approach results do not indicated if updates to the language based on the analysis also improve the language prior introducing it. We think that such an analysis cannot replace the involvement of a real user, but it is useful to identify language concepts that need improvement.

Freire et al. [9] conducted a systematic review on tool support for planning and conducting controlled experiments in general. They found that tool support is mainly limited to the planning phase. The experimenter is often supported by some sort of DSL or ontology for defining an experiment. Support means either textual as presented by Garcia et al. [17] and Siy and Wu [18] or graphically as of Cartaxo et al. [19]. The textual ontologies allow the formal definition of experiments and checking constraints regarding the validity of such.

Freire et al. [20] proposed the first integrated environment supporting all required steps of an experiment, i.e., planning, execution and interpretation. This was done by developing the ExpDSL approach.

Additionally expressibility and completeness was empirically evaluated [11] in various case studies.

Efforts in developing experiment environments have been taken by various researchers. Mostly they differ for their application domains.

In the domain of high-performance computing a set of integrated tools for supporting experiments have been developed by Hochstein et al. [21]. The 'Experiment Manager Framework' is supporting almost all experiment related activities but also has some limitations. Especially the adaptation to a different domain such as DSL engineering is difficult. For example the built-in automated data capturing and its analysis have been developed for high performance computing source code. Other limitations are regarding the lacking support for extending the platform with new statistical design types, as this is not possible easily.

Sjøberg et al. [22] developed SESE (Simula Experiment Support Environment), a web based experiment environment. Subjects are guided through the experiment with assignments. For each assignment the time to spend it, is measured. Additional material can be distributed centrally and the framework allows the centralized collection of results and generated artifacts such as source code. The solution is applicable in many different domains introduces, however, lots of media breaks, as subjects have to switch between applications and tools. As of now, automated metric capturing for language related aspects are not realizable in a web based tool. Thus, using SESE for language related experiments is not optimal.

Because of the limitations with existing approaches and the possibilities on operating within the very same language workbench a language is developed, we have decided to provide an integrated tool environment for DSL experimentation. Instead of reinventing everything, we reused different tested languages and tools. Planning an experiment is done with ExpDSL as introduced above. It forms the base template for our tool chain. ExpDSL differs from a technological point of view, as it is based on Xtext. Our environment bases on MPS.

Our DSL experimentation environment is the first one that supports all phases of experimentation in DSL engineering in one integrated environment. As mentioned before, we presented the integrated tool environment itself, which is evaluated in this paper, already in a previous publication [10]. In addition, in [23] we used the DSL engineering environment to evaluate whether support for business domain concepts is beneficial for DSLs to specify acceptance test cases. In this paper, we provide an evaluation of our integrated DSL experimentation environment based on the technology acceptance model.

3 Running Example Description

With the aid of the integrated tool environment we have performed a DSL experiment for comparing two similar testing languages for acceptance testing of web applications.

The experiment was realized with 22 graduate computer science students in the context of an exercise during a lecture. Especially, for acceptance testing, students are well suited as experimental subjects if a domain familiar to the students is chosen, because then the experiment setting with student participants is similar to the situation of acceptance testing in industry, where it is common to have acceptance tests executed by test personnel with domain knowledge, but only little experience in systematic testing, which is normally the case for students [24]. The domains for defining acceptance tests were therefore selected to be familiar to students, i.e., printing of a signed certificate of studies as well as performing a simple task on the used teaching platform.

An important requirement for the adaptation of a test language is that the time for defining a certain test case is not significantly higher after the adaptation. In reality time can be influenced by many factors such as usability issues or understanding problems. Therefore, such a metric should always be triangulated with other results, e.g., from questions of an accompanying questionnaire. Nevertheless, the analysis of creation time is well suited for describing the tool environment.

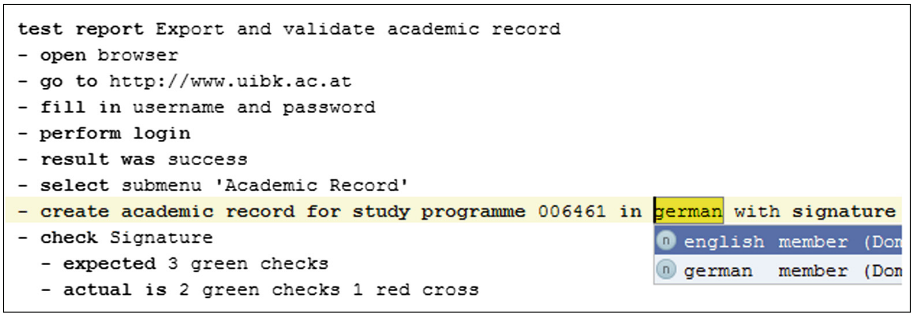
In the real experiment we have also analyzed various other metrics and factors, among them perceived quality of the created test cases.

Listing 1 shows a part of a test case written with the simple DSL (DSL_1). It only consists of two predefined keywords, the step keyword, denoted with "-", and the check keyword.

Listing 1. Example for the simplified test language

```
Test Case Specification: Participate to student
                          seminar
– open a browser
– check whether it has started successfully
– visit http://www.uibk.ac.at
– check if the site has been loaded
– click on the Link named OLAT
...
```

Figure 2 shows a test case with the extended DSL (DSL_2), in a screenshot taken during the conduction of the experiment.



```
test report Export and validate academic record
- open browser
- go to http://www.uibk.ac.at
- fill in username and password
- perform login
- result was success
- select submenu 'Academic Record'
- create academic record for study programme 006461 in german with signature
- check Signature
  - expected 3 green checks
  - actual is 2 green checks 1 red cross
```

Fig. 2. Example for the enriched test language [10]

Both languages are suitable for writing acceptance tests for web applications. DSL_1 allows the user freedom in writing test cases. It may represent a language introduced in a company for providing initial tool support in writing acceptance tests. The second language, i.e., DSL_2, is enriched with extensions from the actual application domain. It could potentially allow faster definition of test cases compared to DSL_1, but without empirical evidence the language engineer is not sure about that. DSL_2 may represent an evolution of DSL_1 for improving tool support in the creation of acceptance test cases.

Although the problem sounds simple and one could intuitively argue that using a language bound to the business domain will for sure allow a user to create faster results than without, this has not to be the case. Language users may feel restricted or confused by certain language concepts, because they are named different than expected. The conduction of a simple experiment can reveal issues with a language before introducing it by giving the language engineer data-driven decision support.

4 Integrated DSL Experimentation Environment

The architecture of the integrated tool environment is outlined in this section together with a short introduction of the methodology. Each experiment step is described with the aid of the example experiment case as presented above. The architecture and the methodology are closely aligned with the steps on planning and executing a controlled experiment as proposed by Wohlin et al. [8]. Those steps (see Fig. 3) are planning, operation, analysis & interpretation, and presentation & package.

Figure 1 shows an overview on the integrated environment. It consists of three major parts, for planning, conduction and analysis. Each part is supported by specific languages and tools. All accessible from within the language workbench.

The planning phase is supported by reusing the experiment definition language (ExpDSL) developed by Freire et al. [11]. ExpDSL was empirically evaluated in 18 experiments. In its second revision the DSL supports almost the entire experiment stack. MetaR, developed by Campagne [12] simplifies data analysis with R and makes it accessible for people that have limited experience in statistics. This might be the case for some language engineers.

In order to collect metrics during the experiment an extension to the MPS platform has been implemented. It supports the experiment operation phase in various aspects and can be configured within the language workbench.

Packaging and reporting is supported by applying the DSL for LaTeX as provided by Völter [13]. It can be used to automatically create reports and pre-filled LaTeX paper templates, which include R-plots and raw data from which one can generate PDF reports.

4.1 Methodology

In this section each supported experimentation phase is described and exemplified based on the sample experiment as introduced in Sect. 3. The methodology follows the procedure as shown in Fig. 3.

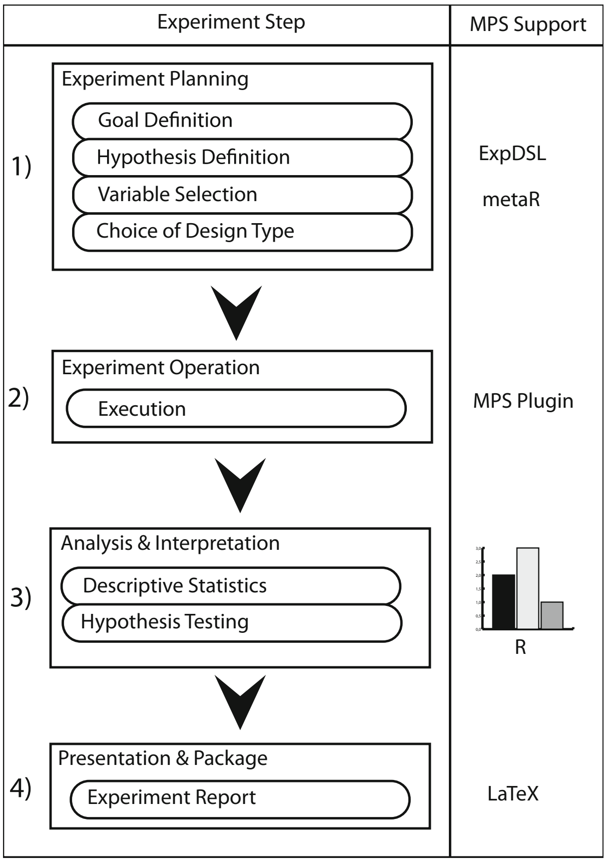


Fig. 3. Supported steps of a controlled experiment [10]

Experiment Planning. For this phase we have reused ExpDSL [11] a DSL for formulating all relevant aspects of an experiment in a precise way. It allows the researcher or language engineer formulating experiment goals, hypotheses, choose a statistical method and to specify dependent and independent variables as well as metrics in a sound way. The base language was extended with features that are based on metaR [12]. Additionally, we have introduced a mapping of certain variables to concrete data, which is collected automatically during the operation phase. This allows an automated generation of certain plots and to automatically test some of the hypotheses.

As shown in the example, some information, e.g., the abstract, is needed for generating the final report.

In practice, a language engineer may only be interested in some basic results of an experiment with lack of full rigor. The basic results are sometimes enough to provide sufficient decision support. Because of that and due to the space limitation we omit details of the complete statistical analysis like power analysis, which are for sure also possible, in the sample experiment description.

The basic example case describes the goals and hypotheses (see Listing 2) together with its alternative. Additionally, Fig. 4 sketches the related hypothesis tests and box plots. In the DSL relevant keywords are highlighted, e.g., 'Goals' or 'Hypotheses'. Additionally the goal question metric template has been implemented [25]. The actual experiment design formulation is a major aspect for the experiment conduction. One not only has to specify statistical methods, but also select variables and their corresponding metrics.

Listing 2. Example goals and hypotheses

```
Abstract This paper presents a controlled experiment...
Goals
G1: Analyze the efficiency of similar test DSLs
for the purpose of evaluation
with respect to creation time
from the point of view of a DSL user
in the context of graduate students using
assisting editors for test case creation.
Hypotheses
H0: The time to create tests with both DSLs is equal
H1: The time to create tests differs significantly
```

Some of the metrics can directly be linked to some of the predefined measures. Such a measure could be for instance the document length, the creation time or also the number of deletions within one language concept. This is enabled because the experiment platform is fully integrated to the actual language workbench.

It is important to mention that as of now, the prototype only supports 'single factor two level' experiment designs. For the upcoming milestone we planned to allow other design types and additional automated language related measurements.

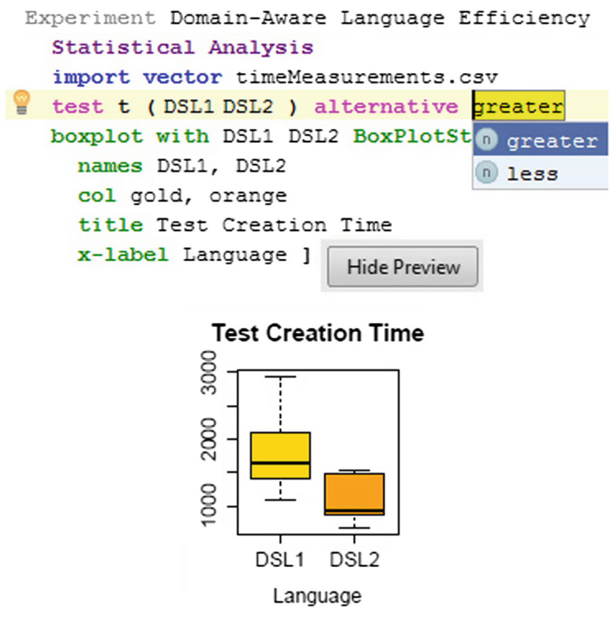
Experiment Operation. In the operation phase, the very same language workbench, used to develop the language and for planning the experiment can be used to conduct the experiment. Basically it has two operation modes, either slave or master (see Fig. 1).

After starting an experiment, test subjects request their assignment from the master instance. While solving the assignment, each client instance collects empirical evidence and sends them back to the master at the end of the experiment. For instance in our case the clients are used by the subjects for creating test cases with either DSL_1 or DSL_2. Among them time measurements, number of deletions and edits.

For supporting the operation phase and enabling the automated collection of data, we have developed an extension plug-in to the MPS language workbench. Each variable as defined in the planning phase is linked to a certain trigger. Start and stop buttons for measuring time in an ongoing task is for example one of those triggers.

The master instance (see Fig. 1) knows the formal experiment description is responsible for distributing assignments and collecting all results. After finishing the experiment, the execution of previously specified statistical analysis procedures is started and a preliminary report is shown. As shown in Fig. 4 it is a modifiable report, which can be adapted to the needs of the language engineer.

Analysis and Interpretation. The analysis & interpretation phase deals with evaluating the hypothesis given collected data. In our case for instance, after performing a normality test, a t-test can be applied in order to verify if the time to create test cases differs significantly.



```
Experiment Domain-Aware Language Efficiency
  Statistical Analysis
  import vector timeMeasurements.csv
  test t ( DSL1 DSL2 ) alternative greater
  boxplot with DSL1 DSL2 BoxPlotSt
    names DSL1, DSL2
    col gold, orange
    title Test Creation Time
    x-label Language ]
```

Fig. 4. Editor within the language workbench with instant preview [10]

Figure 4 shows an example output, that has been generated during the execution of the example experiment as introduced in Sect. 3.

During the analysis one could discover interesting insights. The ability to adapt the analysis on the fly, all assisted by the tool is one of the benefits of having an integrated tool environment.

Figure 4 shows a box plot comparing the creation times of a certain test case for DSL_1 and DSL_2 respectively.

The box-plot already indicates, that the creation time is smaller for DSL_2. For being absolutely sure and verify the significance, one needs to perform further statistical analysis. After performing a normality test, a t-test can be executed. Figure 4 shows the editor right in the moment of the definition of the test. The platform is a very initial prototype supporting a small set of statistical methods. Thus, for detailed analysis there is the possibility to export the raw data as comma separated values (CSV) file. RStudio [26] or RapidMiner [27] allow the import of such data for further analysis.

While using the experiment platform in a real experiment situation, with automated collection of language related metrics, which is important for DSL experimentation, we discovered that there is also a strong need for supporting questionnaires and analysis support.

Questionnaires are not supported as of now, but listed as a top priority feature for the upcoming release.

Presentation & Package. The final phase of the experimentation process is the creation of the report according to to experiment definition. By now the tool environment is able to automatically generate a simple PDF report with information in an easy to read way. Reports are generated from LaTeX templates which base on the recommendations of Wohlin et al. [8]. They contain all relevant information of the experiment. We are aware that this phase consists of more than just a PDF report. Collected data, both raw and generated through analysis, should be included as well as experiment protocols. Until now the prototype is only able to export raw data.

5 Evaluation of the Platform

The integrated experiment platform was used and evaluated in concrete DSL experiments as described in the previous section. In this section we provide an overview of the design of the case study which accompanied the DSL experiments. In the design and reporting of this case study, we follow the recommended guidelines for conducting and reporting case study research in software engineering by Runeson et al. [28].

As already mentioned above we have used the experiment platform for conducting a real DSL experiment. We have found significant differences between two different versions of a language. The actual implementation of the experiment, with the dedicated platform, did not produce any unconsidered issues. The success of the experiment shows already that the tool is useful and suitable for DSL experiments from the viewpoint of a language engineer, as it provides data-driven and evidence-based decision support. Experiments, however, also rely a lot on the commitment of subjects. If they do not find the tool useful, it may bias the outcome and success of the DSL experiment. Therefore we decided to further evaluate the platform from the experiments subject point of view.

The evaluation itself was inspired by the Technology Acceptance Model (TAM) [29]. TAM is a convenient theory used to explain whether, and how, users come to accept new technologies. Basically, it is used to measure the two metrics perceived usefulness and ease of use. Perceived usefulness is defined as

[29] “the degree to which a person believes that using a particular system would enhance his or her job performance”. Perceived ease of use is described as [29] “the degree to which a person believes that using a particular system would be free of effort”.

The subjects of the experiment platform are not going to use the system in their daily job, therefore above mentioned initial definitions are not entirely true for the sake of our evaluation. However both, usefulness and ease of use, as perceived by subjects, can influence the result of an actual DSL experiment. In order to mitigate that risk and keep the bias for the DSL experiment low we have designed the case study as follows.

5.1 Evaluation Design

Based on the goal, question, metric (GQM) template [8,30], we can formulate the goal of this experiment as follows:

In this case study, we investigate whether the domain specific language experiment platform is useful and easy to use with respect to perceived usefulness and perceived ease of use.

Research Questions. From the above formulated GQM template we have identified the following research questions:

- **RQ1:** Is the integrated tool environment useful for creating test case specifications (perceived usefulness)?
- **RQ2:** Is the integrated tool environment easy and intuitive (perceived ease of use)?

Case and Subject Selection. We have developed the experimentation platform for DSL engineering in order to conduct an experiment, in which we compared two different DSLs with respect to various aspects, such as creation time or document length. As part of the DSL experiment, we distributed a questionnaire to the subjects consisting of both quantitative and qualitative questions. Each of the experiment participants was therefore not only a subject in the DSL experiment, but also in this evaluation. Intentionally, the subjects have not been trained before. They only followed a five minute tutorial, where the basic commands of the language editor were demonstrated.

Data Collection Procedure. For the data collection procedure, we prepared a questionnaire that was distributed to the experiment subjects after completing the DSL experiment. There was no time limit for filling out the questionnaire. The questionnaire was filled out on paper and collected at the end of the experiment.

Analysis Procedure. The analysis was performed by two researchers with a mix of qualitative and quantitative methods.

In order to answer both research questions based on answers of the questionnaire, the following concepts have been evaluated as perceived by the experiment subjects. To guarantee that each subject has the same understanding of those concepts, each of them was explained shortly as follows:

To answer RQ1 on perceived usefulness, we based our questionnaire on the work of Davis [31], which elaborated a set of questions, that allow the quantification of perceived usefulness. As already mentioned above perceived usefulness, as originally defined, does not directly apply in our context. Subjects are not going to use the system or tool as part of their daily work. Therefore, we have selected and adapted the questions accordingly (see below). Note that we have not included all TAM related questions. The evaluation is only inspired by TAM.

For answering RQ2 on the perceived ease of use, we based our questionnaire on the System Usability Scale as worked out by Brooke [32]. Again we did not use the entire catalog of questions to keep this side experiment manageable. Similarly to the questions for RQ1, we only did selected those, we thought were relevant for evaluating the ease of use, of the experiment platform.

We are aware, that those adaptation make a comparison to similarly evaluated technology difficult. The resulting trend, however, gives a good indication of both perceived usefulness and ease of use.

- **RQ1.** Perceived usefulness
  - **Support**: Did you feel supported, compared to performing the task in a text editor?
  - **Benefit**: Did you get beneficial support while performing the experiment?
  - **Responsiveness**: Did you have to wait a lot for the tool, while using it?
  - **Restrictiveness**: Did you feel restricted by the tool during the experiment?
- **RQ2.** Perceived ease of use
  - **Intuition**: How quickly do you think other computer scientists need to learn to use this tool?
  - **Usability**: Was the tool in general easy to use?

In the questionnaire, subjects were able to rate above introduced concepts on a four-point Likert-Scale ranging from “Strongly agree” (1) to “Not at all” (4).

For triangulating results from the quantitative evaluation and to get a broader view and understanding on the usability of the experiment platform, we provided a set of open questions. The questions are very general, allowing a lot of freedom and creativity in answering them.

- What did you like about the experiment platform?
- What did you not like about the experiment platform?
- Do you have any suggestions for improvements for the tool?

Answers to those questions have been examined together by two of the authors and after then analyzed using the open coding technique as of Glaser and Strauss [33]. The results to this analysis are presented in Sect. 5.2.

Validity Procedure. Following the guidelines of Runeson et al. [28], the threats to validity of the evaluation have been analyzed according to construct, internal and external validity. In addition, the reliability of the evaluation was also considered. For each type of validity, we have prepared countermeasures. Countermeasures for each validity type are described in detail in Sect. 5.3.

5.2 Results and Discussion

This section presents the results of the evaluation grouped according to the two research questions as introduced in Sect. 5.1. The questions are addressing the perceived usefulness (RQ1) and the perceived ease of use (RQ2). For each of the two research questions, the results are summarized and interpreted. Extreme deviations are discussed in detail.

A total of 22 responses have been collected. As part of the validity procedure and for finding correlations, we have included questions about prior testing experience and DSLs in the questionnaire. Similar to the other quantitative questions subject had to choose from a four-point Likert-Scale. Figure 5 shows the basic statistics on the responses of those questions.

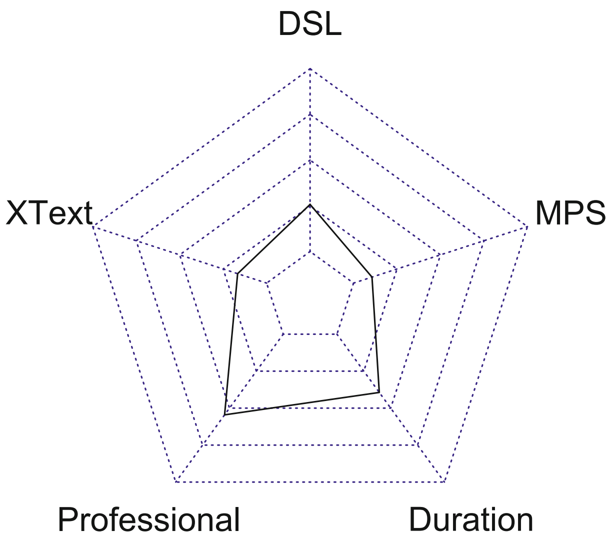


Fig. 5. Prior experience on testing and DSLs

Perceived Usefulness (RQ1). Figure 6 shows the results for the perceived usefulness criteria in form of a violin plot, which is a combination of box plot and density chart.

Subjects were able to answer the questions, as presented in Sect. 5.1, to predefined answers based on a Likert-scale ranging from "Strongly agree" (1) to "Not at all" (4).

For all criteria, the standard deviation was below or around 1, which is an indicator for meaningful results.

The two criteria "Beneficing" and "Restrictiveness" have been perceived as minimal useful. For the latter, this is a desired result, as we wanted the platform support to be as minimalistic as possible. This is good, for keeping variables that are not under the experimenters' control, low.

Initially, we had no explanation for the bad results for the criteria "Beneficing", however, triangulation with open coding revealed the cause. Subjects felt well-supported in the beginning of the experiment, however, they wished, for example to dynamically introduce new language concepts during the experiment. This was definitely out of scope of the experiment, and not intended at all by the experimenters.

The responsiveness of the tool was considered good.

"Support" by the experiment platform, compared to performing the exact same experiment with a regular text editor was evaluated on average as very good. These results indicate, that the participating subjects see the usefulness of the platform, leading to a high perceived usefulness in general.

Perceived Ease of Use (RQ2). The results from the questions as of RQ2 are represented in Fig. 7. Both criteria show that the use of the experiment platform was both intuitive and also easy to use, resulting in a high overall ease of use.

Similarly to the results of RQ1, the standard deviation is very low, in this case for both around 0.6. This indicates meaningful and representative results. Triangulation with open coding as presented later in this section, endorses the results.

Having a good ease of use reduces the learning phase during a DSL experiment and may lower the bias produced by it. In addition, as a great majority of the participating subjects found the environment easy to use, metrics such as time are more accurate and representing the actual time of solving a task, including only little overhead produced by the learning of how to use the tool.

For analyzing the results of the open questions and triangulating the to the quantitative ones, we used the open coding technique as proposed by Glaser and Strauss [33]. Open coding is a repetitive process, in which data is interpretively broken down analytically and assigned to codes. The codes do not belong to any existing theory, they are just based on the actual meaning of the underlying data. This technique is used to get new insights by interpreting phenomena as reflected in data in a non-standard way.

In the following, the open codes, its properties and examples of the participants' exact words are presented for each question as described in Sect. 5.1.

For the question "What did you like about the experiment platform?" we have created the following open codes:

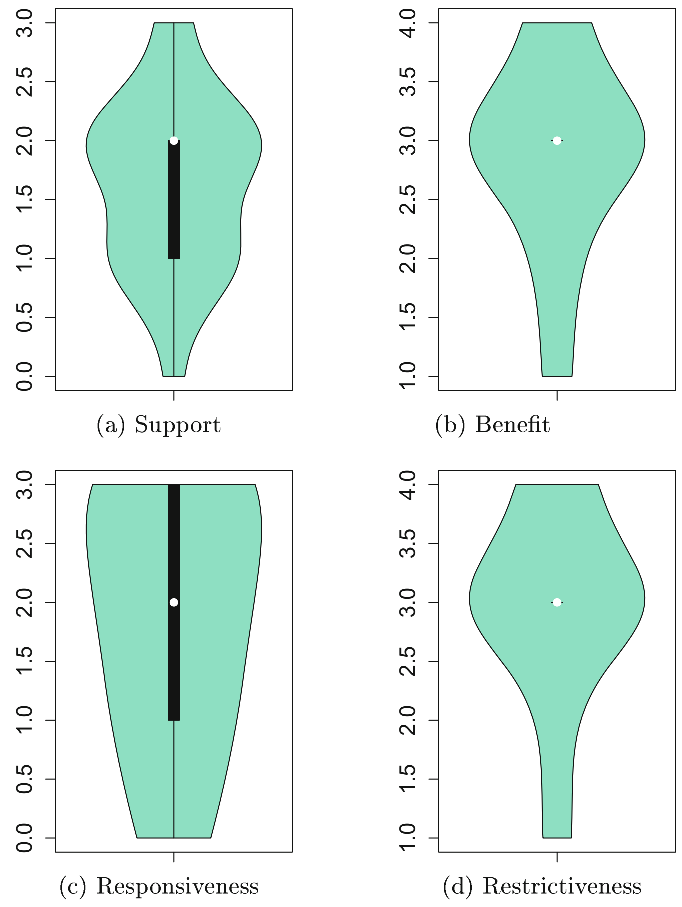


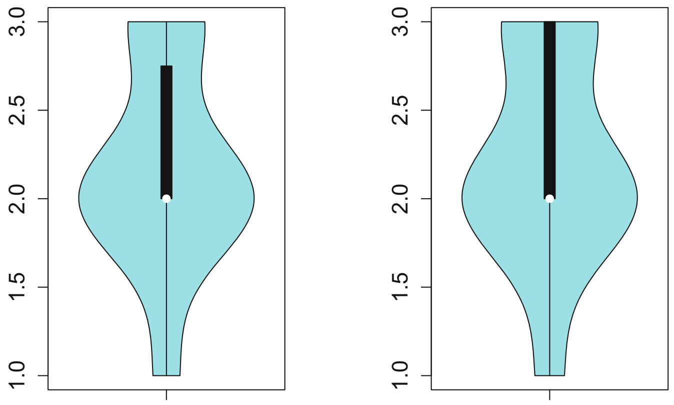
Fig. 6. Results for perceived usefulness criteria

Complexity reducing:

- Even though there were many steps, it was easy to do
- Process is very simple
- Enforces clear structure with a restricted syntax
- The tool provides a good structure, however, it is restrictive

No preliminary knowledge needed:

- Every DSL user should be able to perform the task
- No need to learn the language first
- You only need to know the CRTL+Space shortcut
- Non-computer scientists can learn quickly the tool
- This can be used by a non computer science person as well



(a) Intuition (b) Usability

Fig. 7. Results for perceived ease of use criteria

Simple user interface:

- Simple & easy
- Each step is performed in a simple way
- Intuitive user interface
- Easy to understand

Intuitive usage:

- Straight forward
- The tool is easy to understand and to use
- Each step was intuitive
- Auto-completion feature supports intuitive usage

Quick tool:

- Quickly and easy to write steps
- Fast fluid work-flow

As one can immediately see, results from the open coding coincide with the results of the quantitative analysis. Subjects find the tool very intuitive and easy to use. One additional interesting result, which was not explicitly part of the quantitative analysis, is whether a subject needs some preliminary knowledge prior to performing the experiment task. Open coding revealed that the participants like the intuitive usage and stated that even non-computer scientists can use the tool without learning it before.

Results from the quantitative analysis indicate that the subjects felt restricted by the tool, performing the task. Open coding mitigates that fact. By being restrictive the tool enforces a good and clear structure and reduces the complexity of the outcome.

The answers for the questions "What did you not like about the experiment platform?" and "Do you have any suggestions for improvements for the tool?" only produced one code:

Missing feature:

- Copy&Paste not working
- "Ctrl+Backspace" not working
- No syntax highlighting for some keywords
- Some predefined commands were incomplete

Subjects mainly complained about missing features they know from other tools, or basic features such as copy and paste. This affects the expected perceived usefulness and has to be considered in future releases. In addition, participants did not like that only a small subset of the domain was implemented. Even though that was intended by the experimenter, this also negatively influences the perceived usefulness.

5.3 Threats to Validity

This section presents the threats to validity for our results together with the countermeasures that have been applied to mitigate those threats. As proposed by Runeson et al. [28], they are divided into construct validity, reliability, internal validity and external validity.

Construct validity addresses whether the study really represents answers to the research questions and whether it is consistent with the underlying intention. We wanted to analyze if subjects find the experiment platform easy to use and also useful. Those questions have not only been asked directly, i.e., "Was the tool, in general easy to use?" but were triangulated by additional indirect questions for which open coding was applied to analyze them.

Reliability addresses if the collected and analyzed data can be reproduced by others coming to the same results. It is a well-known threat to all data-centric studies. Results from the open coding analysis technique are of course highly depended on the background and perception of the researcher that are willing to reproduce the evaluation. A counteraction to this fact was that two of the authors did the open coding on their own. After then the results have been compared. We did not find any crucial differences during the comparison, which suggests, that others may come to the same results. In addition, we did prepare a personality test based on the Big Five [34]. It consisted of five questions all related to the conscientiousness. As a result of the test, which was very positive on average, we can ensure the validity of both the qualitative and the quantitative questions. For further increasing the validity, the test was provided in the mother tongue of the subjects, in order to minimize understanding issues.

Internal validity concerns that an observation between treatment and outcome is causal and not as desired a result of factors that are not under the experimenters control. We have considered that information exchange among participants may threaten the internal validity. Therefore we have communicated, that the participants had to work on their own without communicating to other participants in order to mitigate this threat.

External validity is concerned with the fact to what extent it may be possible to generalize the findings. It is a known issue with student experiments that they may threaten the external validity. According to Tichy [35] there are four situations that allow the use of students as subject: (1) if they are well trained for performing the task they are supposed to fulfill, (2) when comparing different methods, the trend of the difference is to be expected to be comparable to that of professionals, (3) in order to eliminate hypotheses, for example if a student experiment reveals no effect on an observable, it this is also very likely to have the same outcome with professionals, or (4) as a prerequisite for experiments with professionals. For the context of our evaluation situations 1–3 apply. This makes it very likely that the subject selection does not influence the external validity. The results can therefore to a reasonable extent be interpreted in a similar way to those of a comparable study performed in an industrial setting. For further lowering the external validity we may have to replicate the evaluation. We think that because of the advantages of projectional over textual editors it may be difficult to generalize those findings also to other language workbenches such as Xtext.

6 Conclusion

This paper presented a fully integrated tool environment for experimentation in DSL engineering based on the Meta Programming System (MPS) language workbench. All steps of experimentation, i.e., planning, operation, analysis & interpretation and presentation & packaging, are supported within the very same environment. Each step is supported by a specialized DSL implemented in MPS or by an MPS extensions, e.g., for automated metrics collection. A language engineer or a researcher is supported in performing experiments on DSLs by providing immediate, evidence-based decision support. The environment is demonstrated with a running example, where two DSLs for acceptance testing of web applications are evaluated with respect to test case creation time. The example shows the practical application of the integrated environment. To the best of our knowledge our approach is the first one providing end-to-end support for experiments in DSL engineering within a single integrated tool environment.

The application of the platform in a real DSL experiment, revealed interesting significant differences between two languages used for testing web applications. This actual application shows already the usefulness from the viewpoint of an experimenter, therefore we decided to empirically evaluate it from the viewpoint of experiment subjects. This not only because the commitment of subjects is essential for experiments. If the tool has a good perceived usefulness and ease of use we can expect a low bias created in learning how to use the tool.

Based on our initial findings we can say that the subjects had, in general, no problems with using the experiment platform in a real DSL experimentation situation. This is not only true for subjects, that had prior experience with DSLs in general or tools such as Xtext or MPS, but also for unexperienced participants. Results from the qualitatively evaluated questions, with open coding analysis, substantiate those finding. Both ease of use and usefulness are showing a positive trend, which may indicate, that the experimentation platform has only little influence on the result of the DSL experiment. However, this finding has to be further evaluated empirically, for instance by conducting case studies and interviews with DSL engineers or other users of our experimentation platform.

We already planned the extension of our tool by integrating the entire feature set as described in ExpDSL. This includes for example the definition and execution of questionnaires. Additionally, one important lesson we have learned during the conduction of the real experiment as described in this paper is the fact that the automatically collected metrics need to be triangulated with qualitative analysis. Open and closed coding analysis appeared to be a suitable analysis method for that issue, thus we plan to implement support for it in the upcoming release. Furthermore, we also plan to conduct an experiment in an industrial setting for improving the tool environment and for evaluating it from the perspective of professionals.

Acknowledgment. This research was partially funded by the research project MOBSTECO (FWF P26194).

References

1. Fowler, M.: Domain-Specific Languages. Addison-Wesley Signature Series (Fowler). Pearson Education, London (2010)
2. Völter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L.C.L., Visser, E., Wachsmuth, G.: DSL Engineering - Designing, Implementing and Using Domain-Specific Languages (2013). dslbook.org
3. Dias-Neto, A.C., Travassos, G.H.: Model-based testing approaches selection for software projects. Inf. Softw. Technol. **51**(11), 1487–1504 (2009)
4. Kosar, T., Bohra, S., Mernik, M.: Domain-specific languages: a systematic mapping study. Inf. Softw. Technol. **71**, 77–91 (2016)
5. Carver, J.C., Syriani, E., Gray, J.: Assessing the frequency of empirical evaluation in software modeling research. In: EESSMod (2011)
6. Gabriel, P., Goulao, M., Amaral, V.: Do software languages engineers evaluate their languages? arXiv preprint arXiv:1109.6794 (2011)
7. JetBrains Team: MPS: Meta Programming System. Jetbrains (2015)
8. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2
9. Freire, M.A., da Costa, D.A., Neto, E.C., Medeiros, T., Kulesza, U., Aranha, E., Soares, S.: Automated support for controlled experiments in software engineering: a systematic review. In: The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, 27–29 June 2013, pp. 504–509 (2013)

10. Häser, F., Felderer, M., Breu, R.: An integrated tool environment for experimentation in domain specific language engineering. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, p. 20. ACM (2016)
11. Freire, M., Kulesza, U., Aranha, E., Nery, G., Costa, D., Jedlitschka, A., Campos, E., Acuña, S.T., Gómez, M.N.: Assessing and evolving a domain specific language for formalizing software engineering experiments: an empirical study. Int. J. Softw. Eng. Knowl. Eng. **24**(10), 1509–1531 (2014)
12. Campagne, F.: MetaR: a DSL for statistical analysis. Campagne Laboratory (2015)
13. Völter, M.: Preliminary experience of using mbeddr for developing embedded software. In: Tagungsband des Dagstuhl-Workshops, p. 73 (2014)
14. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: Evaluating the usability of domain-specific languages. In: Recent Developments, Formal and Practical Aspects of Domain-Specific Languages (2012)
15. Izquierdo, J.L.C., Cabot, J., López-Fernández, J.J., Cuadrado, J.S., Guerra, E., de Lara, J.: Engaging end-users in the collaborative development of domain-specific modelling languages. In: Luo, Y. (ed.) CDVE 2013. LNCS, vol. 8091, pp. 101–110. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40840-3_16
16. Tairas, R., Cabot, J.: Corpus-based analysis of domain-specific languages. Softw. Syst. Model. **14**(2), 889–904 (2015)
17. Garcia, R.E., Höhn, E.N., Barbosa, E.F., Maldonado, J.C.: An ontology for controlled experiments on software engineering. In: SEKE, pp. 685–690 (2008)
18. Siy, H., Wu, Y.: An ontology to support empirical studies in software engineering. In: 2009 International Conference on Computing, Engineering and Information, ICC 2009, pp. 12–15. IEEE (2009)
19. Cartaxo, B., Costa, I., Abrantes, D., Santos, A., Soares, S., Garcia, V.: ESEML: empirical software engineering modeling language. In: Proceedings of the 2012 Workshop on Domain-Specific Modeling, pp. 55–60. ACM (2012)
20. Freire, M., Accioly, P., Sizílio, G., Campos Neto, E., Kulesza, U., Aranha, E., Borba, P.: A model-driven approach to specifying and monitoring controlled experiments in software engineering. In: Heidrich, J., Oivo, M., Jedlitschka, A., Baldassarre, M.T. (eds.) PROFES 2013. LNCS, vol. 7983, pp. 65–79. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39259-7_8
21. Hochstein, L., Nakamura, T., Shull, F., Zazworka, N., Basili, V.R., Zelkowitz, M.V.: An environment for conducting families of software engineering experiments. Adv. Comput. **74**, 175–200 (2008)
22. Sjøberg, D.I., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F., Vokác, M.: Conducting realistic experiments in software engineering. In: 2002 Proceedings 2002 International Symposium on Empirical Software Engineering, pp. 17–26. IEEE (2002)
23. Häser, F., Felderer, M., Breu, R.: Is business domain language support beneficial for creating test case specifications: a controlled experiment. Inf. Softw. Technol. **79**, 52–62 (2016)
24. Felderer, M., Herrmann, A.: Manual test case derivation from uml activity diagrams and state machines: a controlled experiment. Inf. Softw. Technol. **61**, 1–15 (2015)
25. Van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal question metric (GQM) approach. Encycl. Softw. Eng. (2002)
26. RStudio Team: RStudio: Integrated Development Environment for R. RStudio Inc., Boston, MA (2015)
27. RapidMiner Team: Rapid-I: RapidMiner: Predictive Analytics Platform (2015)

28. Runeson, P., Host, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. Wiley, Hoboken (2012)
29. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. **13**(3), 319–340 (1989)
30. Shull, F., Singer, J., Sjøberg, D.I.: Guide to Advanced Empirical Software Engineering, vol. 93. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5
31. Davis, F.D.: User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. Int. J. Man Mach. Stud. **38**(3), 475–487 (1993)
32. Brooke, J.: SUS-a quick and dirty usability scale. Usability Eval. Ind. **189**(194), 4–7 (1996)
33. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Transaction Publishers, Piscataway (2009)
34. Rammstedt, B., John, O.P.: Measuring personality in one minute or less: a 10-item short version of the big five inventory in English and German. J. Res. Pers. **41**(1), 203–212 (2007)
35. Tichy, W.F.: Hints for reviewing empirical work in software engineering. Empirical Softw. Eng. **5**(4), 309–312 (2000)