Information and Software Technology 79 (2016) 52–62

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Is business domain language support beneficial for creating test case specifications: A controlled experiment

Florian Häser*, Michael Felderer, Ruth Breu

University of Innsbruck, Technikerstr. 21a, A-6020 Innsbruck, Austria

ARTICLE INFO

Article history:
Received 27 November 2015
Revised 4 July 2016
Accepted 4 July 2016
Available online 5 July 2016

Keywords:
Domain Specific Languages (DSL)
Behavior driven development
Controlled experiment
Software testing
Student experiment

ABSTRACT

Context: Behavior Driven Development (BDD), widely used in modern software development, enables easy creation of acceptance test case specifications and serves as a communication basis between business- and technical-oriented stakeholders. BDD is largely facilitated through simple domain specific languages (DSL) and usually restricted to technical test domain concepts. Integrating business domain concepts to implement a ubiquitous language for all members of the development team is an appealing test language improvement issue. But the integration of business domain concepts into BDD toolkits has so far not been investigated.

Objective: The objective of the study presented in this paper is to examine whether supporting the ubiquitous language features inside a DSL, by extending a DSL with business domain concepts, is beneficial over using a DSL without those concepts. In the context of the study, benefit is measured in terms of perceived quality, creation time and length of the created test case specifications. In addition, we analyze if participants feel supported when using predefined business domain concepts.

Method: We investigate the creation of test case specifications, similar to BDD, in a controlled student experiment performed with graduate students based on a novel platform for DSL experimentation. The experiment was carried out by two groups, each solving a similar comparable test case, one with the simple DSL, the other one with the DSL that includes business domain concepts. A crossover design was chosen for evaluating the perceived quality of the resulting specifications.

Results: Our experiment indicates that a business domain aware language allows significant faster creation of documents without lowering the perceived quality. Subjects felt better supported by the DSL with business concepts.

Conclusion: Based on our findings we propose that existing BDD toolkits could be further improved by integrating business domain concepts.

1. Introduction

In Behavior Driven Development (BDD) [1,2], which is widely used in modern software development, acceptance tests provide the starting point for the software design flow and serve as a basis for the communication between business-oriented and other stakeholders [3]. BDD is a software development process based on the ideas of test-driven development (TDD) [4]. It combines the principles of TDD on the system level with ideas from domain-specific software engineering [5]. This is because BDD is largely facilitated through the use of a simple domain specific language (DSL) that can express the technical test domain concepts of test behavior as well as the expected outcomes. A DSL for testing helps to convert structured natural language statements into executable tests. Language support is therefore essential for the success of BDD, which relies on the concept of a ubiquitous language from domain-driven design [6]. A ubiquitous language is a (semi-)formal language that is shared by all members of a software development team – both technical and business personnel [6].

In an empirical study by Solís and Wang [7] all commonly used toolkits for BDD were analyzed. The authors found that so far BDD toolkits like Cucumber [2] or RSpec [8] actually do not support the idea of the ubiquitous language. None of the underlying DSLs provides business domain concepts, but all just implement a minimalistic, DSL for BDD that includes the keywords ‘given’, ‘when’, and

* Corresponding author.
E-mail addresses: florian.haeser@uibk.ac.at (F. Häser), michael.felderer@uibk.ac.at (M. Felderer), ruth.breu@uibk.ac.at (R. Breu).

http://dx.doi.org/10.1016/j.infsof.2016.07.001
0950-5849/

'then' to express tests in the form 'Given [initial context], when [event occurs], then [ensure some outcomes]'.

The integration of business domain concepts to implement ubiquitous language features is a promising issue, which enriches the test language. It may improve the quality of test case specifications and productivity of BDD. So far these issues of business domain concept integration into BDD have not been not been empirically investigated. This paper fills this gap and presents an empirical study based on a controlled student experiment that analyzes the perceived quality, the creation time and length of test case specifications, as well as support by the language as perceived by testers.

The objective of the study presented in this paper is to examine whether supporting ubiquitous language features inside a DSL, by extending the DSL with business domain concepts, is beneficial over using a DSL without those concepts. In the context of the study, benefit is measured in terms of perceived quality, creation time and length of the created test case specifications. In addition, we analyze if participants feel supported when using predefined business domain concepts.

The controlled experiment was conducted with graduate students at the University of Innsbruck based on a novel platform for DSL experimentation that has been implemented for that purpose. In the experiment, the students had to create test case specifications in order to test two teaching platforms used by the students on a regular basis. This business domain is familiar to the students and they are therefore well suited as experimental subjects for specifying test cases created with either a DSL with business concepts or a DSL without business concepts in a test-driven way.

The paper is structured according to the guidelines of Jedlitschka and Pfahl [9] for reporting controlled experiments in software engineering.

The remainder of this paper is organized as follows. In Section 2 we introduce the underlying DSL concepts for BDD. Then in Section 3, related work and its relation to this study is presented. In order to enable the reproducibility of the controlled experiment, Sections 4 and 5 describe in detail the experimental design and the execution. The results are provided in Section 6 and then interpreted in Section 7. Finally, Section 8 concludes the paper and presents an outlook to future work and for further analysis.

2. DSLs for behavior driven development

In TDD, requirements are specified as tests and after implementation transformed into executable test cases to evaluate their proper implementation and acceptance. Code Listing 1 shows an example test scenario in which the store owner wants to make sure that returned items are put back into stock. The very same example was given to the subjects in the lecture before the experiment when introducing DSLs.

As mentioned in the previous section, BDD is usually implemented in a domain-specific language (DSL). Which is a high-level software implementation language that supports concepts and abstractions that are related to a particular (application) domain [10]. For instance, for testing the DSLs UML Testing Profile [11] as well as Telling TestStories [12] have been defined and successfully applied for model-based testing.

However, analyzing the supported language concepts from Listing 1 reveals that those concepts are very generic and only from the testing domain. The supported concepts, written in bold, includes the keywords 'story', 'in order to', 'as a', 'I want to', 'scenario', 'given', 'and', 'when', and 'then'. It allows the tester to express tests in the form 'Given [initial context], when [event occurs], then [ensure some outcomes]' including a description, postcondition and a requirement that is to be verified. Besides the keywords the tester is allowed to write everything in form of free text.

In reality however, while writing the test case specification he should be supported with concepts from both the vending (business) and the testing (technical) domain. Examples for business domain concepts are 'stock' and 'item' including their corresponding operations, such as 'add' and its semantic. The business domain aware toolkit only allows the addition of items to the stock and ensures thereby high quality right from the creation of the test case specification.

For our experiment we have created a very simple DSL with fewer language concepts than provided by BDD. It only consists of a 'step' (denoted by '–') and a 'check' step language concept. An example test case specification created with this language is shown in Listing 2. The listing represents an excerpt of the test case specification created with the simple DSL during the trial run. Note that limiting the DSL has no impact on the expressibility of the test scenario. Only the support a tester gets by the tool environment when creating the test is limited. In the experiment, as described later in the paper that language is built into a DSL without support for business domain artifacts, i.e. without including ubiquitous language features.

The language presented in Fig. 1 represents a test language tailored to the actual business domain that is to be tested. The screenshot was taken during the experiment trial run and represents a part of the test case specification. The DSL includes domain concepts from various domains. When designing the language we have included domain concepts from the execution environment, in our case the web browser. The tester is supported by concepts like 'open', 'go to', 'fill in' etc. Furthermore, we also considered the actual business domain. The web application allows the user to perform several tasks. We have analyzed some of them and included them into our DSL. The language concept 'create academic record' for example takes two parameters, the study program identifier and the language of the resulting document. Both parameters have a limited set of selection possibilities. The web applications allows only two languages, English and German. The DSL and the editor that supports the tester knows those selection alternatives and proposes them while writing the test scenario (see Fig. 1).

While all DSLs, as presented in this section, are suitable for creating test case specifications, in order to test web applications, only the latter one is aware of the actual domain. The language supports and guides the tester not only on the technical, but also on the business level.

3. Related work

In this section we present related empirical studies to DSLs and testing.

Kosar et al. [13] provide a comprehensive systematic overview on the entire research field of DSLs. The overview also includes empirical validations of such. They confirmed an observation of Carver et al. [14] that within the DSL community, researchers are much more interested in creating new techniques and languages, than performing empirical evaluations.

In addition, researcher focus mostly on the design and implementation, rather than domain analysis and maintenance, which are also important for DSL engineering. Evidence is provided, which show that DSLs are rarely validated by the end-user and researchers usually assume that the developed DSLs is perfectly tailored for the domain. Recent publications deal with that downside by, for instance providing a usability evaluation framework for DSLs, as elaborated by Barišić et al. [15]. Other approaches, as of Izquierdo et al. [16], involve the real end-user in the entire development process for building and evaluating a DSL. In contrast to that, Tairas and Cabot [17] present a very interesting post-hoc

```
Story: Returns go to stock

In order to keep track of stock
As a store owner
I want to add items back to stock when they're returned

Scenario 1: Replaced items should be returned to stock
Given that a customer buys a blue garment
And I have two blue garments in stock
And three black garments in stock.
When he returns the garment for a replacement in black,
Then I should have three blue garments in stock
And two black garments in stock
```

Listing 1. Example for the BDD 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
...
```

Listing 2. Example test case specification for the simple DSL.

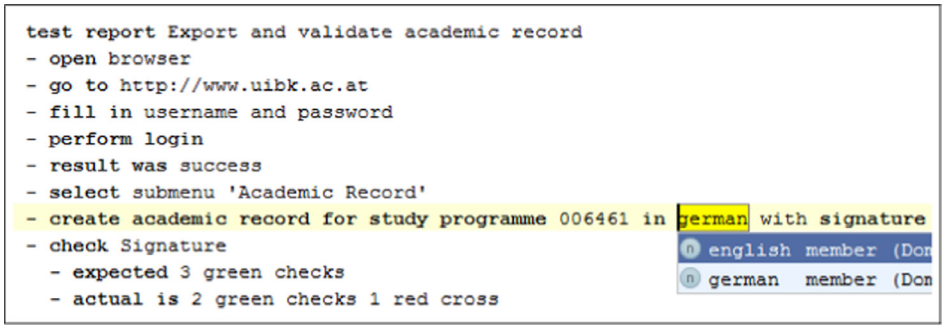


Fig. 1. Example output for the DSL with business concepts.

evaluation analysis. In their methodology a DSL is analyzed based on its actual use after it has been deployed. This may give the language engineer a good overview on business domain artifacts that may need improvement. Although they evaluated their approach it is not said, whether the actual inclusion of identified new domain concepts may also improve the language.

Acceptance Test Driven Development (ATDD) as introduced in the previous Section 1 is usually implemented and provided through a DSL, providing concepts from the testing domain. ATDD and the more general TDD have already been intensively empirically analyzed. George et al. [18] for example realized a set of structured experiments involving 24 professional pair programmers on test driven development. They have investigated, whether tests written a priori or posteriori are more effective. Effectiveness was measured in time for coding all tests and as number of errors found. They found that developers are slower, but find more errors when using the test driven methodology.

Erdogmus et al. [19] conducted a similar controlled experiments with undergraduate students. While one group followed the test-first strategy, the control group implemented with a more conventional test-last development technique. Both groups had to follow an incremental process, adding new features at a time and imme-

diately test them. They have decided not to measure time as productivity indicator. Basic findings of the study are that with test-first on average more tests have been written. Quality was the same regardless of the development technique. A minimum linear quality increase was observed with the number of tests. Fucci and Turhan [20] replicated the study and empirically confirmed the results of the study.

Madeyski published an entire book [21] including several controlled experiments on test first solo development over test last solo development. He discovered that the test first practice does not have a statistically significant impact on various things e.g. the number of acceptance tests passed or number of acceptance tests passed per development hour (as development time indicator). The only interesting property that really differs was code coupling. With test first developers produced code that was significantly less coupled. Such code is usually more reusable and maintainable.

Several empirical studies, for instance by George and Williams [18], Gupta and Jalote [22], as well as Janzen et al. [23], provide evidence that TDD helps to improve software quality.

The benefits of DSLs over other kinds of representation, such as code or text have also been investigated empirically before. Usually empirical evidence is collected in the very specific domain the DSL is targeted for. For example Kuzniarz et al. [24] and Ricca et al. [25] conducted a controlled experiment on whether using stereotypes really improves the understanding of UML models. They have used a graphical representation of UML in their experiment, the results however, are partially generalizable for all types of DSLs.

Kosar et al. [26] conducted a series of experiments on the comprehension of DSLs over general purpose programming languages. They observed that the comprehension took significantly less time, i.e. correctness and efficiency is higher when using DSLs. Although the experiment was not related to testing and the compared language was a general purpose programming language the results of the research is quite interesting and related.

The framework for integrated tests (Fit) can be introduced post launch with the intention of maintaining the products quality. Because it is designed to be a testing language for arbitrary business domains, it includes however just a limited set of domain artifacts that are related to testing. The real application domain is not reflected in the Fit tables. In their research Ricca et al. [27] investigate the advantages of using the Fit for requirement changes during maintenance over textual representation of the requirements. The research showed that the overhead of using the framework is negligible. Additionally, the correctness of the maintained code is much higher. Extreme programming, i.e. working in pairs nearly compensated the benefits of Fit when it comes to correctness but not for the time aspect.

Language aspects where not investigated in the literature so far. Above mentioned studies only partially answer our research objectives. They only provide some hints, whether the base assumption for our framework, i.e. incrementally including business domain concepts into a testing language, as presented in Sections 1 and 2 might work as expected.

4. Experimental design

In this section, we discuss planning and execution of the experiment, which include hypotheses, variables, experiment design, subjects, objects and instrumentation. In addition, we explain the data collection, analysis and validity procedures.

4.1. Problem statement

Given the case that we have, two similar languages, one very general including a small set of keywords, named L_G and the other one including business domain concepts, miming the ubiquitous language concept of BDD in this publication specified by the term L_B. Both languages are well suited for writing acceptance tests for web applications. As already mentioned the second language, i.e. L_B, however has been enriched with business domain concepts that potentially allow a faster and clearer definition of tests in that language, compared to L_G. Our assumption is that the language that includes concrete domain knowledge, i.e. L_B is superior to the DSL with business concepts L_G.

4.2. Research objectives and questions

Based on the goal, question, metric (GQM) template [28,29], we can formulate the goal of this experiment as follows:

Analyze the output of a DSL with business concepts and a DSL without them *for the purpose of* evaluation
with respect to their perceived quality, creation time, document length and support *from the point of view of the* DSL user
in the context of master students writing acceptance test case specifications.

4.3. Hypotheses and variables

The goal of the experiment is to investigate the advantages on supporting business domain concepts over a DSL without those business concepts. This is achieved through various measurements.

From the GQM template and the problem statement, we can derive the following research questions.

- **(RQ1)** Is the perceived quality of test case specifications created with L_B higher than the ones created with L_G?
- **(RQ2)** Do creation time or specification length differ significantly from each other?
- **(RQ3)** Do participants feel more supported by a language that contains business domain concepts?

4.3.1. Independent variables

The independent variable in the experiment is the language type, i.e. L_G (DSL for testing without business concepts) and L_B (DSL for testing with business concepts).

L_G: Is a very simple DSL that includes two concepts: test step and check step. Each action has to be formulated in free text. The subject has total control over the test case specification creation. As use case we have asked the subjects to log in to the teaching system, select a course and apply for a certain seminar topic. After receiving a confirmation mail they were asked to search the Wikipedia article for this topic, save the page as PDF document and submit it to the teaching system. Finally they had to check, whether they received the confirmation mail.

L_B: The language implemented for this treatment is based on the Fit [30]. We have extended the base concepts from the language to include also features that are crucial for web testing. That includes actions like 'go to page'. Moreover, we have added language elements and options from the domain of the use case. In this treatment the subjects had to log in onto the self-service area of the student administration. Thereafter they were asked to export their academic record in English, including only positive marks. The resulting PDF document had to be digitally signed. In order to verify the validity of the signature the document had to be uploaded to the official Austrian governmental signature verification service .[1] Finally the verification report should be checked in order to have a valid document.

[1] https://www.signatur.rtr.at/en/elsi/Pruefung.

4.3.2. Dependent variables

For answering the three research questions the following dependent variables have been defined and measured. Research question **RQ1** (as stated in Section 4.1) on the quality of test case specifications that are created by each of the different DSLs is answered by analyzing the outcome based on several criteria. In the experiment we have observed the following dependent variables, with respect to the used language. All criteria have been measured by subjectively rating them.

The empirical evaluation method is based on the framework for empirical evaluation of model comprehensibility as proposed by Aranda et al. [31]. The assessment relies on a four-point Likert-Scale ranging from "Completely Agree"(1) to "Not at all" (4). Lozano et al. [32] revealed that the number of alternatives is between four and seven. We have decided to include four alternatives, i.e. without the neutral element, in order to get a more concrete trend between agreement and disagreement.

We have chosen a crossover design for measuring perceived quality of the resulting test case specifications. A specification sheet was given randomly to a subject of the other group and assessed through a questionnaire. Each criterion corresponds to one question in the questionnaire and results in a value ranging from 1 to 4.

Perceived support was measured similarly by the subjects themselves.

Kahraman and Bilgen [33] proposed a framework for qualitatively assessing DSLs. The framework is based on the ISO/IEC 25010:2011 standard and defines a set of quality characteristics that should be considered when creating a DSL. Those characteristics are functional suitability, usability, reliability, maintainability, productivity, extendability, compatibility, expressiveness, reusability and integrability. We did not want to evaluate the test language itself but only the output. Therefore we concentrated on the expressiveness, usability and productivity characteristics. We have selected the following criteria to evaluate them.

- **Completeness:** Is the test scenario complete, e.g. in terms of all relevant test steps, alternatives and expected failures.
- **Correctness:** Are all steps that have been documented in the test case specification correct.
- **Comprehensibility:** Is the test case specification easy to understand.
- **Reading flow:** Because of the shorthand notation in DSLs the reading flow may be disturbed leading to a bad readability.
- **Reproducibility:** Can someone repeat the acceptance test based on the document assuming that this person has similar skills and is aware of the domain?

For **RQ2** we measure the time it takes to create the test case specification. In addition to time, we also consider the document length, based on the number of test steps.

Research question **RQ3** on the support of a language that includes business domain concepts, is qualitatively and quantitatively investigated through a questionnaire including open questions. After a series of internal discussions we have selected the following criteria, for measuring perceived support.

- **Benefit:** Rates whether there is a generated benefit in creating the test scenario with the tool over writing it freely in a document.
- **Intuitiveness:** Used to determine if the approach and tool can be used with little training.
- **Restrictiveness:** Did the subject feel restricted by using the predefined language concepts?
- **Supportive:** Was the approach supporting you in creating the test case specification?
- **Usability:** Rate the usability of the approach and tool.

In the open questions we have asked the subjects what they liked and disliked when performing the assignment and their suggestions for improvement.

4.3.3. Hypotheses

Based on those criteria we came up with the hypotheses as follows, were c is a variable for the criteria as defined above. The specific hypotheses were then tested individually on each criterion:

- Hc^{i}_{null} There is no significant difference for the criteria when using L_G vs using L_B.
- Hc^{i}_{alt} The use of a language significantly differs for the measured criteria.

4.4. Subjects

As subjects for the experiment we have selected a group of 22 graduate students that attended a lecture on advanced topics in software engineering. 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 [34,35]. The domains for defining acceptance tests where 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.

4.5. Objects

Based on the problem statement, reported in Section 4.1, we implemented a prototype [36], which allows the assisted rapid creation of test case specification. The prototype itself, as one can see in Fig. 2 is based on the DSL workbench MPS [37]. Furthermore, we have enriched the workbench with plug-ins in order to automatically collect measurements. Moreover we have removed all features of the workbench that were not relevant for the experiment in order to prevent distraction and interference. The experiment itself has been specified in a DSL suitable for experiment definition, including goals, hypotheses and metrics. All data has been collected on the experiment master and exported as comma separated version file for further analysis. For evaluating perceived quality and perceived support we relied on a questionnaire, which is provided from our website .[2]

4.6. Instrumentation

In the lecture before the experiment students have been introduced to DSLs.

Students were randomly separated into two groups and rooms. The examiner introduced to both separated groups to the exact same experiment task, i.e. the same story:

> "Imagine the situation that you as domain expert, have to test a feature that gets an update within the next month. Unfortunately you are on vacation on the planned release. Therefore you write down every relevant test step, such that your substitute can use it as a base for verifying the correct functionality of the new feature."

After then the students have been introduced to the basics of the tool via a short tutorial. The tutorial was given by the examiner on the beamer and followed step by step by the students. They

[2] http://qe-informatik.uibk.ac.at/experiment-material-dsl-business-concepts/.

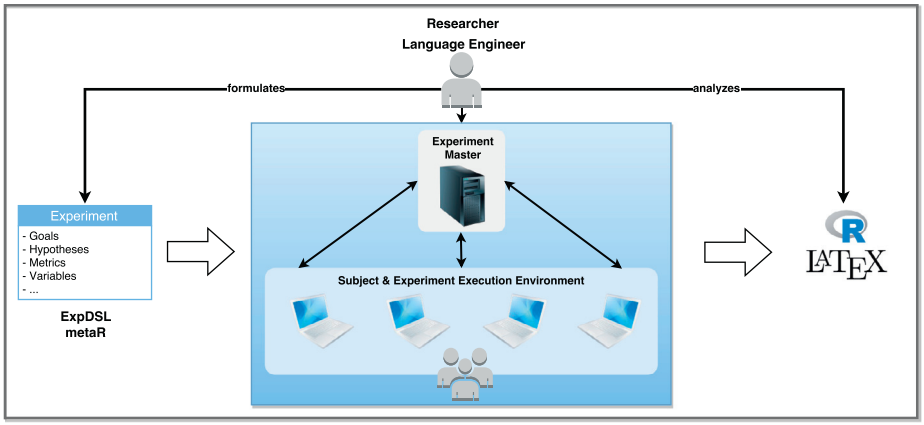


Fig. 2. Experiment platform.

Table 1
Experiment Schedule.

5 min	Introduction & random separation of two equally sized groups
5 min	Moving of one group to separated room
20 min	Story & Step-by-step tutorial with examiner
45 min	Execution of the assignment
10 min	Questionnaire 1 on background, personality and experiment impression
10 min	Synchronization of groups & printing of test case specifications
5 min	Random distribution of the test case specifications in the other group
10 min	Quality evaluation of the test case specifications through Questionnaire 2

have been shown how new documents are created and how to use the editor, especially the auto complete and suggestion features.

4.7. Data collection procedure

This subsection describes the experimental schedule and the data collection procedures.

In order to answer all the research questions and to minimize the threats to validity we have used several methods for data collection. Among different questionnaires with quantitative and qualitative questions we also relied on automatic measurements collected by the experiment environment.

Table 1 shows the experiment schedule, including allowed time and task.

Data for RQ1 was collected in Questionnaire 2 by randomly distributing a test case specification to a subject from the other test group and letting it subjectively evaluated the test case specification according to the quality criteria. Moreover, the subject had to find out the use case represented by the test case and summarize it in at most five sentences. We used the summary for mitigating the threat of misinterpretation. The data for RQ2 was collected and linked to the other measurements by the experiment environment during the execution of the assignment. Each subject was asked to press on a start button whenever they feel ready to perform the assignment and press on a stop button as soon they finish. Questionnaire 1 contained questions, whether the subjects felt supported by either of the languages and open questions on what they disliked and what they liked on the test methodology. The data was used to answer RQ3.

In order to analyze both languages L_G and L_B we have prepared two treatments. Basically, the assignment in each of the treatments was the creation of a test case specification, using an assisting editor, for a certain use case. As we wanted the subjects to be domain experts, we have chosen the use cases from their everyday interaction with the teaching system [3] and the self-service student management system .[4]

4.8. Analysis procedure

The statistical analysis procedure and the resulting plots as later presented in the paper have been performed and created with the aid of statistical computing platform R [38].

One of our objectives was to analyze the time and length of the test case specification created with the different languages L_G and L_B. We have applied the Shapiro–Wilk normality test to determine whether both, time and length are normally distributed. If this is the case, it is legit to test H_2 by means of the two-tailed paired parametric t-test. Otherwise, if the corresponding data is not normally distributed, we apply the non-parametric two-tailed Wilcoxon signed rank test.

The perceived quality of test case specification is measured through the criteria as discussed in Section 4.3. This procedure allows the immediate application of the non-parametric two-tailed Wilcoxon rank test [31] for validating the null hypotheses.

For each hypothesis test we use a significance level of $\alpha = 0.05$. In addition, for all results that differ significantly, the effect size and statistical power is calculated with the G*Power Statistical Power Analyses [39] software tool. As significance level β we choose 4-times α ($\beta = 0.2$), as recommended by Cohen [40]. The results are then described in Section 6 and discussed in Section 7.

4.9. Evaluation of validity

We base our planned evaluation of the validity on the recommendations of Wohlin et al. [29]. They propose four types: conclusion, internal, construct and external validity. All the precautions are discussed in detail in the section about the threats to validity 7.2.

5. Execution

In this section, we discuss the actual execution of the experiment. It was conducted according to the schedule shown in Table 1 and already discussed in Section 4. In this section, we discuss the sample, preparation, performed data collection and validity procedure.

[3] https://lms.uibk.ac.at/dmz/.
[4] https://orawww.uibk.ac.at/public/lfuonline.login.

5.1. Preparation

The students have been divided into two groups. Each group received a different treatment. We have selected the balanced design approach [41], i.e. each treatment is having an equal number of students as experimental subjects. The separation of the students into the different groups was performed randomly.

The experiment editor was distributed to the students via the network drive. Each student copied the editor onto his computer and opened the tutorial project. The tutorial was presented by the examiner with the aid of the beamer and followed by the students step-by-step. Students were allowed to interrupt at any time in order to ask questions.

Thereafter the task has been presented and a short background (see Section 4.6) has been told in order to get comparable results.

5.2. Performed data collection

The assigned task, was the creation of a test case specification, using an assisting editor, for a certain use case. For this purpose, we prepared two treatments as described in detail in Section 4.3.

The data collection was performed with two questionnaires and with the experiment tool. The exact timing is shown in Table 1.

5.3. Validity procedure

Regarding the preparation process we mitigated the risks to validity by letting just one examiner giving the tutorial and introducing the task. Both groups took the very same almost word-by-word tutorial. The participants were said that they are doing the task for practicing their knowledge on DSLs. Moreover, they have been told that the outcome will not be assessed by the teacher. During the synchronization break we asked the subjects not leaving the room or at least not to speak to the other group.

6. Results

In this Section the summary of the collected data and a detailed description of the analysis are presented. The data needed for answering our research questions as defined in Section 4.2 has been collected through different means. As mentioned in Section 4.7 we prepared two questionnaire, one purely quantitative, the other including qualitative questions and automatic measurements, collected by our experiment platform. In total, we have gathered 22 measurements from 22 subjects, i.e. 11 per group. The data for answering each of the research questions RQ1 to RQ3 is first presented in a descriptive manner. Afterwards, the hypotheses, as constructed in Section 4.3, are validated according to the analysis procedure introduced in Section 4.8.

6.1. Descriptive statistics

For each of both treatments as mentioned in Section 4.3 we have collected a total of 11 individual measurements. The perceived quality, as questioned in **RQ1**, was subjectively evaluated by the subjects, based on the different quality criteria. The result is shown in Table 2. We have encoded the responses from the questionnaire to numeric as presented in the result tables according to the scheme “Completely Agree”(1), “Agree”(2), “Disagree”(3), “Completely Disagree”(4). No criterion received the worst assessment. Table 2 shows means and standard deviations, grouped by language L_G and L_B. Comparing them, and by having a look at the box plots, reveals already that the perceived quality is almost the same, i.e. it does not differ significantly. Similarly is the case with the length, as questioned by **RQ2**, of the test case specifications. The descriptive statistics of time for creating the specification and its length in test steps are shown in Table 3.

Mean and standard deviation for the variables answering **RQ3** are shown in Table 4. While the intuitiveness and the usability are rather similar, the generated benefit and supportive show expected results.

6.2. Hypothesis testing

We followed the analysis procedure as described in Section 4.8. In order to select the correct statistical test, a distribution test had to be executed whenever needed.

RQ1: By strictly following the analysis procedure 4.8 the distribution test can be omitted and the Wilcoxon rank test immediately applied. As shown in Table 2, for all of the quality criteria the p-value shows no significant difference. That means H_0 cannot be rejected.

A graphical representations in form of box-plots is shown in Fig. 3. Perceived quality is defined as the sum of each quality criterion. We have considered each criterion to be equally important.

RQ2: For selecting the correct statistical test we checked the distribution of creation time and document length with the Shapiro–Wilk normality test. A significance level of $\alpha = 0.05$ was defined. P-values of 0.08047 for the time measurements and of 0.1588 for the document length were calculated. For both time and length H_0 cannot be rejected, hence, the data is normally distributed. This allows the application of the t-Student test for validating the hypotheses. Document length has a p-value of 0.2937 and H_0 cannot be rejected. That means the document length produced by the subject with both treatments do not differ significantly.

In contrast to that, the t-test of the document creation time delivers a p-value of 0.004186. Effect size and power calculations were performed in a post-hoc analysis with the appropriate parameters in G*Power. As statistical test we did select the two tailed difference between two independent means, as we used the t-test for calculating the p-value. The observed effect size was $d = 1.4278923$ and the corresponding statistical power $1 - \beta = 0.8895535$. The results show a large effect and an acceptable power, hence H_0 can be rejected and the alternative hypotheses H_1 holds.

The test case specification creation is significantly faster when using the language that is extended with business domain aspects.

The results for **RQ3** have been analyzed similarly as in RQ1. For each criterion as introduced in Section 4.3 the Wilcoxon rank test was applied. Only for one of these criteria, “supportive” a p-value of 0.01056 was calculated, which is within the acceptable range. Power and effect size have been post-hoc calculated with the appropriate parameter in G*Power (two tailed Wilcoxon–Mann–Whitney test for two independent means). The resulting effect size is $d = 1.2789316$ and a power value of $1 - \beta = 0.7939808$ was calculated. While the resulting values show a large effect, the power is below the acceptable $\beta = 0.2$, because of the small sample size. Hence H_0 cannot be rejected.

7. Interpretation

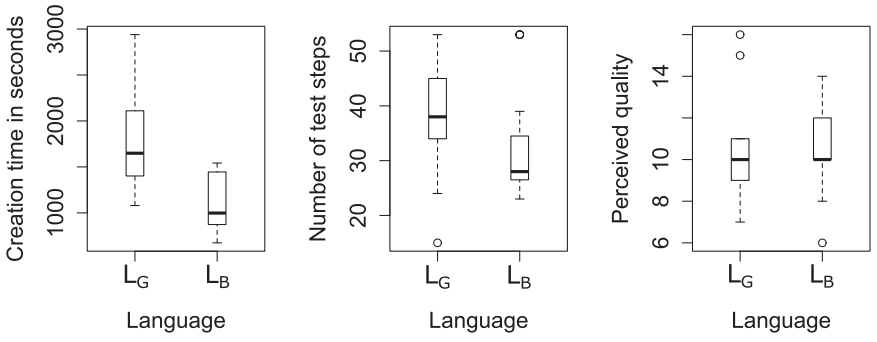
In this Section after evaluating and interpreting the results we discuss the limitations of this study in form of threats to validity. Finally, we conclude by sharing our experience gathered while conducting the study.

7.1. Analysis of results and implications

Including business domain concepts, i.e. supporting the ubiquitous language features inside a DSL, significantly decreases the

Table 2
Mean (M), standard deviation (SD), median (Mdn) and p-values of the quality criteria.

	L_G			L_B			p-value
	M	SD	Mdn	M	SD	Mdn	
Completeness	1.55	0.52	2	1.36	0.50	1	0.4245
Correctness	1.64	0.81	1	1.45	0.52	1	0.7396
Comprehensibility	1.64	0.67	2	1.82	0.60	2	0.4829
Reproducibility	1.45	0.69	1	1.55	0.52	2	0.5748
Reading Flow	2.09	0.83	2	2.55	0.82	3	0.2479



(a) Creation time (b) Test steps (c) Perceived quality

Fig. 3. Box-plots for creation time, test steps and perceived quality.

Table 3
Mean (M), standard deviation (SD), median (Mdn) and p-values on length and time of test case specifications.

	L_G		L_B		p-value
	M	SD	M	SD	
Time	1769.09	579.55	1099.55	322.27	0.004186
Length	37.82	11.29	32.73	10.85	0.2937

creation time, while increasing the subjective feeling on being supported by the toolkit and predefined language constructs was observed. This result is similar to the findings as discussed in the related work section. Kosar et al. [26] did empirically compare DSLs and general purpose programming languages (GPLs). They have observed that with DSLs the time for performing a task was significantly lower than with GPLs. We assume that the significant decrease in time, in our experiment, is due to the fact that the subjects are not forced to think about a suitable formulation when writing the test case specification. Instead subjects get an acceptable sentence in which they just have to enter the correct parameters. The assumption is also covered by the result that participants felt more supported by the L_B language. Initially, we also assumed that the perceived quality is significantly higher in L_B over L_G. In reality, however, it is almost the same for both languages. This is also true for related studies in a similar field. Even though, a direct comparison with similar quality observations, from related TDD studies, is not that feasible, they indicate a similar trend. Erdogmus et al. [19] revealed in a study that the quality is not influenced by the test first or test last technique. A replication of the study conducted by Fucci and Turhan [20] confirmed the observation.

Thus one could argue that quality is highly influenced by the real user regardless the applied tool or method in general.

The personality test showed a high conscientiousness value for all subjects, which supports the quality considerations. Without knowing the concrete quality criteria, they all tried to create a comprehensible and reproducible test case specification. Although, only introduced verbally, they have tried to follow the instruction as described in Section 4.6 "...write down every relevant test step..." It shows that the results from the personality test are meaningful.

The participants felt better supported by the ubiquitous language features inside the DSL. The perceived support results may be influenced by two factors, the tool support and the language support. Since RQ3 targets the actual language support, we have planned different countermeasures for reducing the risk of measuring tool support. A control question was included questioning the perceived restrictiveness regarding the predefined language. Results show a higher restrictiveness for L_B which is an expected result and indicates that the perceived support question was understood correctly and is language related. Additionally,

Table 4
Mean (M), standard deviation (SD), median (Mdn) and p-values of perceived support.

	L_G			L_B			p-value
	M	SD	Mdn	M	SD	Mdn	
Benefit	2.45	0.82	2	1.82	0.75	2	0.08847
Intuitiveness	1.82	0.40	2	1.82	0.75	2	0.9078
Supportive	2.82	0.60	3	2.09	0.54	2	0.01056
Usability	1.64	0.50	2	1.73	0.65	2	0.8197
Restrictiveness	1.55	0.60	2	1.91	0.30	2	0.0675

triangulation with the qualitative questions further mitigates this threat. The students were asked to concisely write down what they liked and what they disliked in the approach. For L_G, an 8 of 11 participants disliked the fact that one had to think about every test step and its formulation. On the other hand they liked the possibility to formulate everything on their own. Seven students with the L_B treatment liked that some essential things have been already included to the language. However, they disliked that not every aspect of the entire domain is already included. 4 of 11 subjects wanted a feature in order to be able to model "if-then-else" constructs. An additional functionality, requested by 3 participants, was the possibility to define domain concepts on their own and reusing them during the assignment. This is very similar to the creation of methods in general purpose programming languages.

Nevertheless, we can claim that supporting the ubiquitous language features inside a DSL, by incrementally enriching the base language could be considered for future BDD implementations.

7.2. Limitations of study

In this section we discuss certain issues, which may have threatened the validity of the experiment presented in this paper. Similar to the evaluation of validity as presented in Section 4.9 we consider conclusion, internal, construct and external validity [29].

Conclusion validity is concerned with data collection, the relationship between treatment and outcome and the reliability of the measurement. Our experiment was done by 22 participants. This is a rather small critical size that may not provide enough data to derive significant results. For making sure that there is a statistical, significant relationship between the treatment and the outcome we have planned some measures before conducting the experiment. The questionnaires have been proofread and filled out by several colleagues. Each question or aspect that was unclear, was immediately corrected or enriched with an explanation or a small example. The assessment of the quality criteria, essential for answering RQ1 was performed by the experiment subjects and is highly subjective. We were aware of the risk of subject judgments and included several measures to mitigate them. The subjects were asked to summarize the test case specification in maximal five sentences. Those summaries can then be randomly checked by the examiner in order to validate the comprehension and the validity of the judgment. Wherever possible we have prepared objective metrics, as time and document length. Additionally, we have prepared a personality test based on the Big Five [42]. Only a small subset, for finding out the conscientiousness of the students was used. The test was provided in the mother tongue of the subjects, for minimizing understanding issues. The outcome of the personality test showed that each of the subjects is very reliable and can be trusted in the subjective rating.

We had no issues with the random heterogeneity of subjects, which is usually a threat for controlled experiments and the conclusion validity. Our subjects as described in Section 4.4 are recruited from group of graduate students, all having a very similar background. In the questionnaire we asked for prior testing and DSL experiences.

In addition to the p-value, effect size and statistical power have been calculated for verifying if H_0 was rejected correctly.

Internal validity concerns that an observation between treatment and outcome is causal and not a result of a factor that is not under the experimenters control or measurement. We considered that an observation between treatment and outcome is not the result of a factor by several means. Each group has to be introduced to the experiment and guided through the tutorial by the same experimenter. Information exchange among participants and groups may threaten the internal validity. Subjects within the same group should have enough space in the room for not interacting with each other. We communicated that the participants had to work on their own and that they should not communicate with the other group during the synchronization phase. No subject received special instructions. If something was not understood, it was not explained to the subject exclusively but to the entire group. In Questionnaire 1 we checked for any issues that may have occurred with the actual tool.

Construct validity checks whether the experiment setting is actually reflecting the research objective. Our primary focus, was to show which of the approaches is better. We have clearly defined what better means. It is measured through document length, perceived quality and creation time. Quality again was defined as measurable metrics. Those metrics and measurements that are collected for answering the research questions have been defined by the authors of this paper in several meetings on basis of an empirical framework for the evaluation of model comprehensibility [31], related studies [34,35] and on the possibilities of measurements. The subjects had enough time to answer all questions and to perform the experiment tasks. A control group of two participants performed the tasks prior the experiment, for getting a feeling for time and the measurements.

Another threat for the construct validity is the evaluation apprehension of the subject, i.e. participants are afraid of being evaluated and may lie. The students were not graded by the teacher. They were told that the experiment is just an exercise to practice the use of DSLs. Moreover, we have eliminated the time pressure, by planning a large enough time frame. We did not define any explicit "quality measure" measuring the test case specifications quality with one single value. Such a metric is questionable as quality is always constructed with quality criteria. In the evaluation of the other test case specification, for verifying that the students have understood the use case, we asked them to summarize the test case specification in maximal five sentences. We randomly checked those summaries in order to validate how good the subject understood the test case specification. We have not found any discrepancies.

The experiment tool was kept very simple, with minimal functionality, such that the students were able to focus on the experiment assignment. We have conducted a trial run with two students for setting up the schedule and for getting an initial feeling, whether we are observing the right thing. Those students did not participate in the main experiment, as they would probably have introduced a bias.

External validity is concerned with to what extent it is possible to generalize the findings. In order to evaluate if both DSLs differ significantly with respect to the research objective, it should ideally be applied in an industrial context by professionals. Approaches like the one presented in the context of the paper are however often evaluated in experiments performed with students. It is a known issue with student experiments that they may threaten the external validity.

According to Tichy [43] four situations are acceptable to use students as experimental subject: (1) if students are trained well enough to perform the task they are asked for, (2) when comparing methods, the trend of the difference can be expected to be comparable to that of professionals, (3) to eliminate hypotheses, i.e. if in a student experiment no effect is observable this is also very probable with professionals, or (4) as a prerequisite for experiments with professionals. In our case all of the situations 1–3 apply. This means that selecting students as samples is not threatening our external validity. Our experiment setting is similar to the situation of acceptance or system testing as performed in industry. There it is common to have system or acceptance tests performed by participants with domain knowledge, and little experience in systematic testing [34,35]. The required testing skills are provided in short trainings, which is similar to the tutorial

phase in our experiment. As long as the students are aware of the domain, we think that students and professionals are comparable. The simplicity of the experiment and the limited number of subjects however, do not allow a generalization of the results to an industrial setting of arbitrary complexity.

7.3. Lesson learned

During the experiment we did not encounter any problem. The students followed the experiment schedule as planned. A very crucial thing was the trial run of the experiment performed with two participants. Based on that trial we were able to exactly schedule the experiment and to get an initial feeling for the possible results. A significant part of the questionnaires was devoted to ensure the validity of the study, e.g. by a psychological personality test based on the Big Five [42]. In future experiments, we try to minimize this overhead to a minimum in favor of increasing the sample size.

8. Conclusions and future work

In this paper, we empirically evaluated, whether supporting the ubiquitous language characteristics as specified in BDD compared to using the base language. We conducted a controlled student experiment with 22 participants and observed differences in perceived quality of the resulting test case specifications, time to create such a specification and whether participants feel supported by business domain concepts compared to specifications written using the base language. The domain was well known by the participants and the setting is therefore comparable to professionals.

We observed that the inclusion of business domain concepts allows significantly faster creation of test case specifications, while document length and quality is not significantly affected. The participants felt slightly more restricted with predefined sentences, however they also felt more supported by such an approach.

We claim that supporting business domain concepts could improve existing BDD toolkits.

In our previous research we proposed and implemented a framework [44] that is similar to BDD, but applied in a different context. Our framework, enables a non-intrusive introduction of (model-based) testing by supporting the tester in documenting tests that are manually executed. A possibility to equip BDD with the appropriate business domain concepts, could be implemented by an iterative learning process like the corpus-based analysis of DSLs as proposed by Tairas and Cabot [17]. Identified constructs are formalized iteratively in an assisted learning process. Each formalized language construct can then be implemented and linked to executable test code and thus allowing (semi-)automated execution of acceptance tests.

Because of the promising results achieved, we plan to continue that direction of research. The limited number of subjects and the simplicity of the experiment do not allow a generalization of the results. Thus, we will perform a replication of the controlled experiment to refine our findings and for enabling generalization. Finally we plan an additional case studies in an industrial context and to provide and evaluate guidelines for domain element enrichment for testing languages in practice.

Acknowledgment

This research was partially funded by the research projects QE LaB - Living Models for Open Systems (FFG 822740) and MOBSTECO (FWF P26194). In addition, we thank all participants of the experiment for their time and effort.

References

[1] D. North, Introducing BDD, Dan North & Assosciates, 2015. http://dannorth.net/introducing-bdd/
[2] M. Wynne, A. Hellesoy, The Cucumber Book: Behaviour-Driven Development for Testers and Developers, O'Reilly, 2012.
[3] M. Soeken, R. Wille, R. Drechsler, Assisted behavior driven development using natural language processing, in: Objects, Models, Components, Patterns, Springer, 2012, pp. 269–287.
[4] K. Beck, Test-driven development: by example, Addison-Wesley Professional, 2003.
[5] B.R. Bryant, J. Gray, M. Mernik, Domain-specific software engineering, in: Proceedings of the FSE/SDP workshop on Future of software engineering research, ACM, 2010, pp. 65–68.
[6] E. Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional, 2004.
[7] C. Solś, X. Wang, A study of the characteristics of behaviour driven development, in: Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on, 2011, pp. 383–387.
[8] D. Chelimsky, D. Astels, B. Helmkamp, D. North, Z. Dennis, A. Hellesoy, The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends, O'Reilly, 2010.
[9] A. Jedlitschka, D. Pfahl, Reporting guidelines for controlled experiments in software engineering, in: Empirical Software Engineering, 2005. 2005 International Symposium on, IEEE, 2005, pp. 95–104.
[10] E. Visser, Webdsl: A case study in domain-specific language engineering, in: R. Lämmel, J. Visser, J. Saraiva (Eds.), Generative and Transformational Techniques in Software Engineering II, Lecture Notes in Computer Science, volume 5235, Springer Berlin Heidelberg, 2008, pp. 291–373.
[11] P. Baker, Z.R. Dai, J. Grabowski, I. Schieferdecker, C. Williams, Model-driven testing: Using the UML testing profile, Springer Science & Business Media, 2007.
[12] M. Felderer, P. Zech, F. Fiedler, R. Breu, A tool-based methodology for system testing of service-oriented systems, in: Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on, IEEE, 2010, pp. 108–113.
[13] T. Kosar, S. Bohra, M. Mernik, Domain-specific languages: A systematic mapping study, Inf. Softw. Technol. 71 (2016) 77–91.
[14] J.C. Carver, E. Syriani, J. Gray, Assessing the frequency of empirical evaluation in software modeling research., EESSMod, 2011.
[15] A. Barišić, V. Amaral, M. Goulão, B. Barroca, Evaluating the usability of domain-specific languages, Formal Prac. Aspects Domain-Specific Lang. (2012).
[16] J.L.C. Izquierdo, J. Cabot, J.J. López-Fernández, J.S. Cuadrado, E. Guerra, J. de Lara, Engaging end-users in the collaborative development of domain-specific modelling languages, in: Cooperative Design, Visualization, and Engineering, Springer, 2013, pp. 101–110.
[17] R. Tairas, J. Cabot, Corpus-based analysis of domain-specific languages, Softw. Syst. Model. 14 (2) (2015) 889–904.
[18] B. George, L. Williams, A structured experiment of test-driven development, Inf. Softw. Technol. 46 (5) (2004) 337–342. Special Issue on Software Engineering, Applications, Practices and Tools from the ACM Symposium on Applied Computing 2003
[19] H. Erdogmus, M. Morisio, M. Torchiano, On the effectiveness of the test-first approach to programming, IEEE Trans. Softw. Eng. 31 (3) (2005) 226–237.
[20] D. Fucci, B. Turhan, On the role of tests in test-driven development: a differentiated and partial replication, Empir. Softw. Eng. 19 (2) (2014) 277–302.
[21] L. Madeyski, Test-Driven Development: An Empirical Evaluation of Agile Practice, Springer Science & Business Media, 2009.
[22] A. Gupta, P. Jalote, An experimental evaluation of the effectiveness and efficiency of the test driven development, in: Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on, 2007, pp. 285–294.
[23] D.S. Janzen, C. Polytechnic, S.L. Obispo, H. Saiedian, Does test-driven development really improve software design quality, IEEE Softw. (2008) 77–84.
[24] L. Kuzniarz, M. Staron, C. Wohlin, An empirical study on using stereotypes to improve understanding of uml models, in: Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on, IEEE, 2004, pp. 14–23.
[25] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, M. Ceccato, How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: a series of four experiments, Softw. Eng. IEEE Trans. 36 (1) (2010) 96–118.
[26] T. Kosar, M. Mernik, J. Carver, Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments, Empir. Softw. Eng. 17 (3) (2012) 276–304.
[27] F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, P. Tonella, Using acceptance tests as a support for clarifying requirements: A series of experiments, Inf. Softw. Technol. 51 (2) (2009) 270–283.
[28] F. Shull, J. Singer, D.I. Sjøberg, Guide to Ddvanced Empirical Software Engineering, volume 93, Springer, 2008.
[29] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, 2012.
[30] R. Mugridge, W. Cunningham, Fit for Developing Software: Framework for Integrated Tests, Pearson Education, 2005.
[31] J. Aranda, N. Ernst, J. Horkoff, S. Easterbrook, A framework for empirical evaluation of model comprehensibility, in: Proceedings of the International Workshop on Modeling in Software Engineering, IEEE Computer Society, 2007, p. 7.

[32] L.M. Lozano, E. García-Cueto, J. Muñiz, Effect of the number of response categories on the reliability and validity of rating scales, Methodology 4 (2) (2008) 73–79.
[33] G. Kahraman, S. Bilgen, A framework for qualitative assessment of domain-specific languages, Softw. Syst. Model. (2013) 1–22.
[34] M. Felderer, A. Herrmann, Manual test case derivation from uml activity diagrams and state machines: A controlled experiment, Inf. Softw. Technol. 61 (2015) 1–15.
[35] M. Felderer, A. Beer, B. Peischl, On the role of defect taxonomy types for testing requirements: Results of a controlled experiment, in: Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, IEEE, 2014, pp. 377–384.
[36] F. Häser, M. Felderer, R. Breu, 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, in: EASE '16, ACM, 2016, pp. 20:1–20:5.
[37] JetBrains Team, MPS: Meta Programming System, Jetbrains, 2015. https://www.jetbrains.com/mps/.
[38] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2015. http://www.R-project.org.
[39] F. Faul, E. Erdfelder, A. Buchner, A.-G. Lang, Statistical power analyses using g* power 3.1: Tests for correlation and regression analyses, Behav. Res. Methods 41 (4) (2009) 1149–1160.
[40] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, L. Erlbaum Associates, 1988. https://books.google.co.in/books?id=TI0N2lRAO9oC
[41] R.E. Kirk, Experimental Design, Wiley Online Library, 1982.
[42] B. Rammstedt, O.P. John, Measuring personality in one minute or less: A 10-item short version of the big five inventory in english and german, J. Res. Personality 41 (1) (2007) 203–212.
[43] W.F. Tichy, Hints for reviewing empirical work in software engineering, Empir. Softw. Eng. 5 (4) (2000) 309–312.
[44] F. Häser, M. Felderer, R. Breu, Test process improvement with documentation driven integration testing, in: The International Conference on the Quality of Information and Communications Technology, 2014. QUATIC 2014. Proceedings, 2014, pp. 156–161.