An Empirical Study to Evaluate a Domain Specific Language for Formalizing Software Engineering Experiments

Marília Freire[a,b], Uirá Kulesza[a], Eduardo Aranha [a], Andreas Jedlitschka[c],
Edmilson Campos [a,b],Silvia T. Acuña [d], Marta N. Gómez [e]
[a] Federal University of Rio Grande do Norte, Department of Informatics and Applied Mathematics, Natal, Brasil
[b] Federal Institute of Rio Grande do Norte , Natal, Brasil
[c] Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany
[d]Universidad Autónoma de Madrid, Madrid, Spain
[e]Universidad San Pablo-CEU, Madrid, Spain
{marilia.freire,edmilsoncampos}@ppgsc.ufrn.br,{uira,eduardo}@dimap.ufrn.br, andreas.jedlitschka@iese.fraunhofer.de,
silvia.acunna@uam.es, mn.gomez@usp.ceu.es

***Abstract*—The research about the formalization and conduction of controlled experiments in software engineering has reported important insights and guidelines for conducting an experiment. However, the computational support to formalize and execute controlled experiments still requires deeper investigation. In this context, this paper presents an empirical study that evaluates a domain-specific language proposed to formalize controlled experiments in software engineering. The language is part of an approach that allows the generation of executable workflows for the experiment participants, according to the statistical design of the experiment. Our study involves the modeling of eight software engineering experiments to analyze the completeness and expressiveness of our domain-specific language when specifying different controlled experiments. The results highlighted several limitations that affect the formalization and execution of experiments. These outcomes were used to extend the evaluated domain-specific language. Finally, the improved version of the language was used to model the same experiments to show the benefits of the improvements.**

I. INTRODUCTION

The conduction of controlled experiments in software engineering has increased over the last years [1] [2]. Experiments and their replication are considered essential to produce scientific evidences and to enable causal effect analysis, improving the knowledge about the benefits and limitations of new and existing software engineering (SE) methods, theories and technologies. In addition, it also promotes the sharing of this knowledge inside the SE community. Consequently, controlled experiments can accelerate the development and evaluation of effective scientific innovations produced by the academy or the industry.

Over the last decade, the community discussed how to better support the application of controlled experiments in the SE research area. These studies have proposed guidelines to report controlled experiments [3], conceptual frameworks to guide the replication of controlled experiments [4], and environments/tools to support their conduction and replication [5] [6] [7]. Although these studies brought important insights and outcomes related to the conduction of controlled experiments, few of them [7][8] [9] [10] [11] have proposed to formalize the planning, execution and analysis of controlled experiments. In addition, the existing approaches that aim at formalizing the specification of controlled experiments – such as ontologies or domain-specific languages – and/or at providing support for their execution and analysis have not been evaluated. Therefore, this subject still requires deeper investigation.

In this context, this paper describes an empirical study conducted to evaluate a domain specific language that provides support to the modeling and execution of SE controlled experiments proposed by our research group [8] [12]. In our study, we have modeled several controlled experiments with the objective of analyzing the completeness and expressivity of the domain-specific language (DSL) and supporting environment of our approach. These criteria were analyzed based on the specification of eight different experiments and considering fundamental experimental aspects documented by the experimental software engineering community [1] [2] [3]. We present the results of our analysis and illustrate how they have been used to improve the evaluated domain-specific language.

This article is organized as follows. Section II describes the study settings. Section III discusses the evaluation results, as well as the new extensions proposed for the evaluated DSL. Sections IV and V present, respectively, the threats to validity and related work. Finally, Section VI presents conclusions and directions for future work.

II. STUDY SETTINGS

This section presents the study settings in terms of: its main goal and research questions (Section II.A), the approach being evaluated (Section II.B), the study methodology (Section II.C), and the evaluation criteria (Section II.4).

A. *Study Goal and Research Question*

Table 1: Experiments modeled in our study.

Experiment	Institution-Country
Testing [17]	UFPE-Brazil
Human Factors [14]	UAM-Spain
Requirements [15]	UPM-Spain
SPrL [20]	UFRN-Brazil
MDD	UPV-Spain
SPL [21]	PUC-Rio-Brazil
CFT [18]	Fraunhofer-IESE-Germany
PBR [19]	University of Maryland-USA

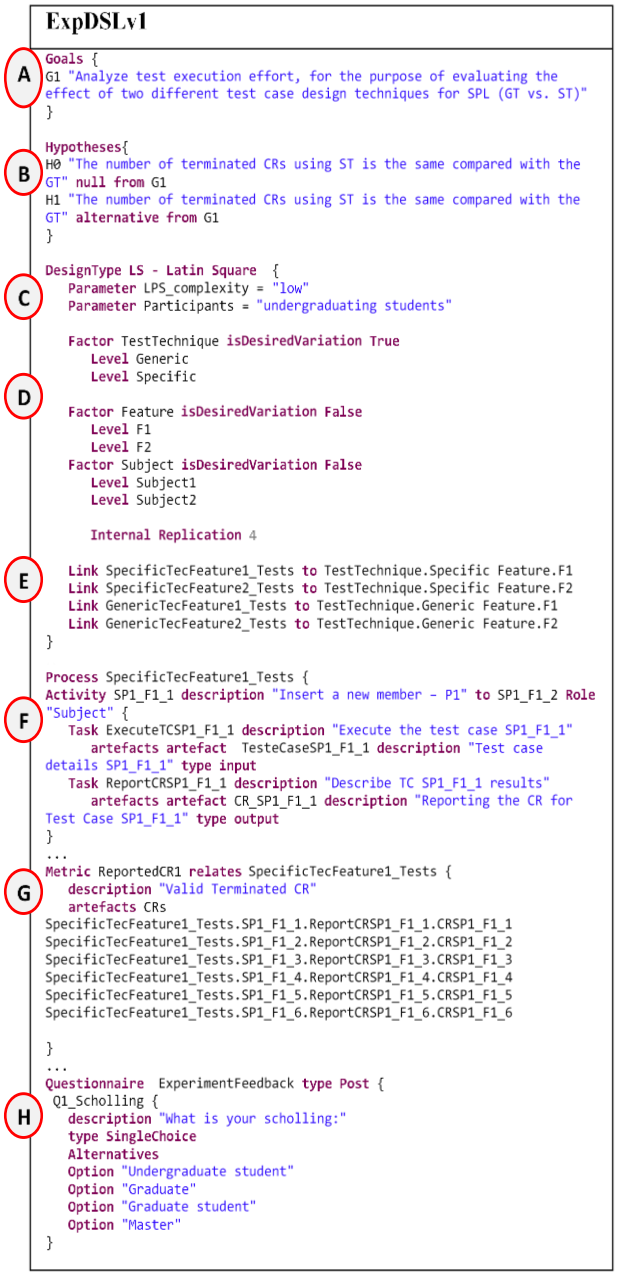
The main goal of this study is to validate the experimental domain specific language with respect to the modeling of SE controlled experiments from the perspective of the experimenters. To achieve this goal, a research question (RQ) was defined: Are the DSL abstractions of the approach adequate to model the different aspects of SE controlled experiments?

In order to answer this question, our study investigated how the different specification aspects of controlled experiments could be addressed by our DSL. Different and complementary criteria were adopted to analyze the completeness and expressiveness of the domain-specific language during the modeling of a set of existing experiments.

B. *Evaluated Domain Specific Language*

Our study focuses on the assessment of a DSL that is part of a model-driven approach for supporting the formalization and execution of experiments in software engineering [8] [12]. The approach consist of: (i) a DSL, called ExpDSL, used to describe the process and statistical design of controlled experiments; (ii) model transformations that allow the generation of workflow models that are specific for each experiment participant and according to the experiment's design; and (iii) a workflow execution environment that guides and monitors the participant activities during the experiment execution, including the gathering of participants feedback during this process.

ExpDSL is a textual DSL that is composed of four main parts/views: process view, metrics views, experimental plan view, and questionnaire view. Each view allows defining the experiment aspects as follows: (i) *Process View* – allows defining the activities of data collection from the experiment participants. It is similar to a software process language, where we can define the activities, tasks, artifacts, and roles involved in the process; (ii) *Metric View* – defines the metrics that have to be collected during the experiment execution. Each metric intercepts process activities or tasks in order to quantify observations of interest (dependent variables, etc.) in the study; (iii) *Experimental Plan View* – defines the experimental plan by identifying the factors to be controlled (treatments, noise variables, etc.) and the statistical design to be used (treatments, participants and experimental material allocation); and (iv) *Questionnaire View* – defines questionnaires to collect quantitative and qualitative data from participants of the experiment. These questionnaires can be applied before or after all the activities of the experiment process.

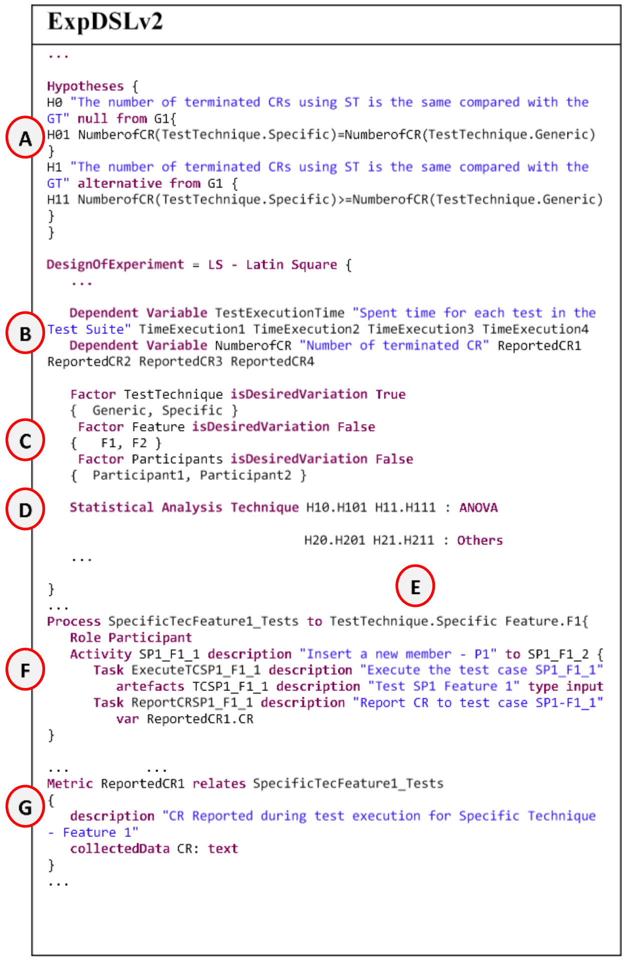


ExpDSLv1

```
Goals {
G1 "Analyze test execution effort, for the purpose of evaluating the
effect of two different test case design techniques for SPL (GT vs. ST)"
}

Hypotheses{
H0 "The number of terminated CRs using ST is the same compared with the
GT" null from G1
H1 "The number of terminated CRs using ST is the same compared with the
GT" alternative from G1
}

DesignType LS - Latin Square  {
   Parameter LPS_complexity = "low"
   Parameter Participants = "undergraduating students"

   Factor TestTechnique isDesiredVariation True
      Level Generic
      Level Specific

   Factor Feature isDesiredVariation False
      Level F1
      Level F2
   Factor Subject isDesiredVariation False
      Level Subject1
      Level Subject2

      Internal Replication 4

   Link SpecificTecFeature1_Tests to TestTechnique.Specific Feature.F1
   Link SpecificTecFeature2_Tests to TestTechnique.Specific Feature.F2
   Link GenericTecFeature1_Tests to TestTechnique.Generic Feature.F1
   Link GenericTecFeature2_Tests to TestTechnique.Generic Feature.F2
}

Process SpecificTecFeature1_Tests {
Activity SP1_F1_1 description "Insert a new member - P1" to SP1_F1_2 Role
"Subject" {
   Task ExecuteTCSP1_F1_1 description "Execute the test case SP1_F1_1"
      artefacts artefact  TesteCaseSP1_F1_1 description "Test case
details SP1_F1_1" type input
   Task ReportCRSP1_F1_1 description "Describe TC SP1_F1_1 results"
      artefacts artefact CR_SP1_F1_1 description "Reporting the CR for
Test Case SP1_F1_1" type output
}
...
Metric ReportedCR1 relates SpecificTecFeature1_Tests {
   description "Valid Terminated CR"
   artefacts CRs
SpecificTecFeature1_Tests.SP1_F1_1.ReportCRSP1_F1_1.CRSP1_F1_1
SpecificTecFeature1_Tests.SP1_F1_2.ReportCRSP1_F1_2.CRSP1_F1_2
SpecificTecFeature1_Tests.SP1_F1_3.ReportCRSP1_F1_3.CRSP1_F1_3
SpecificTecFeature1_Tests.SP1_F1_4.ReportCRSP1_F1_4.CRSP1_F1_4
SpecificTecFeature1_Tests.SP1_F1_5.ReportCRSP1_F1_5.CRSP1_F1_5
SpecificTecFeature1_Tests.SP1_F1_6.ReportCRSP1_F1_6.CRSP1_F1_6

}
...
Questionnaire  ExperimentFeedback type Post {
 Q1_Scholling {
   description "What is your scholling:"
   type SingleChoice
   Alternatives
   Option "Undergraduate student"
   Option "Graduate"
   Option "Graduate student"
   Option "Master"
}
```

Figure 1: ExpDSL fragment before extensions

Fig. 1 shows the fragments of the "Testing" experiment specification [13] using the original version of our DSL (ExpDSLv1). It is a controlled experiment that compares two different black-box manual test design techniques: a generic technique and a product specific technique. The study evaluates those techniques from the point of view of the test execution process. In the specification, we can see an Experimental Plan View (Fig. 1- A,B,C,D,E), a Process View fragment (Fig. 1 – F), a Metric View fragment (Fig. 1 - G) and a Questionnaire View fragment (Fig. 1 – H). Section III presents and discusses those fragments in the context of our study.



```
...

Hypotheses {
H0 "The number of terminated CRs using ST is the same compared with the
GT" null from G1{
H01 NumberofCR(TestTechnique.Specific)=NumberofCR(TestTechnique.Generic)
}
H1 "The number of terminated CRs using ST is the same compared with the
GT" alternative from G1 {
H11 NumberofCR(TestTechnique.Specific)>=NumberofCR(TestTechnique.Generic)
}
}

DesignOfExperiment = LS - Latin Square {
   ...

   Dependent Variable TestExecutionTime "Spent time for each test in the
Test Suite" TimeExecution1 TimeExecution2 TimeExecution3 TimeExecution4
   Dependent Variable NumberofCR "Number of terminated CR" ReportedCR1
ReportedCR2 ReportedCR3 ReportedCR4

   Factor TestTechnique isDesiredVariation True
   {  Generic, Specific }
    Factor Feature isDesiredVariation False
   {   F1, F2 }
    Factor Participants isDesiredVariation False
   {  Participant1, Participant2 }

   Statistical Analysis Technique H10.H101 H11.H111 : ANOVA

                                  H20.H201 H21.H211 : Others
   ...

}
...
Process SpecificTecFeature1_Tests to TestTechnique.Specific Feature.F1{
   Role Participant
   Activity SP1_F1_1 description "Insert a new member - P1" to SP1_F1_2 {
      Task ExecuteTCSP1_F1_1 description "Execute the test case SP1_F1_1"
         artefacts TCSP1_F1_1 description "Test SP1 Feature 1" type input
      Task ReportCRSP1_F1_1 description "Report CR to test case SP1-F1_1"
         var ReportedCR1.CR
}

...          ...
Metric ReportedCR1 relates SpecificTecFeature1_Tests
{
   description "CR Reported during test execution for Specific Technique
- Feature 1"
   collectedData CR: text
}
...
```

Figure 2: ExpDSLfragment after extensions

The experiment modeled in ExpDSL is submitted in our approach to a set of transformations (model to model and model to text) to generate workflow models for the participants according to the statistical design of experiment (DoE). Finally, these workflow models can be executed in a workflow engine (web application), guiding and monitoring the participants during the experiment. For additional details of our approach, please refer to [12].

C. Study Methodology

Our study was organized in three main phases: (i) the selection and specification of different experiments using the evaluated version of ExpDSL herein called ExpDSLv1; (ii) the evaluation of each modeled experiment through the study criteria (Section II.D); (iii) the analysis, discussion, and proposal of improvements for the ExpDSL and the approach considering the study results.

We selected 2 quasi-experiments and 6 controlled experiments with different statistical designs, executed and documented by the software engineering community. We have also considered experiments from different research groups and software development areas (requirements, model-driven, software product lines, testing, human factors). Finally, the selection also took into consideration the availability of information about the experiment planning and conduction. Table 1 lists the experiments used in our study. The specification of the experiments in ExpDSL is available at: http://migre.me/hqP6N. Our team – the authors of this paper – modeled different aspects of the experiments using the ExpDSLv1 based on their available documentation. For most of the experiments, we have also interacted with the researchers that conducted them.

In the last phase of our study, we analyzed the collected results for each specified experiment, and evaluated how the completeness and expressiveness criteria were tackled by the approach. During this analysis, we investigated for each criterion: (i) the reasons for the (non)adequate specification of each specific aspect of the experiment; and (ii) which improvements can be applied to the DSL to address a better modeling of the identified aspects. Finally, we implemented these new improvements and re-modeled the same experiments using the improved version of ExpDSL (herein called ExpDSLv2) to highlight the achieved results. Fig. 2 shows fragments of new or modified elements in ExpDSLv2.

D. Adopted Criteria

Our study adopted three main assessment criteria used for evaluating DSLs [13]:

Completeness: All concepts of the domain can be expressed in the DSL.

Expressiveness: The degree to which a problem solving strategy can be mapped into a program naturally. In particular in this study, we evaluated the orthogonality aspect of expressiveness [13] that states that each DSL construct is used to represent exactly one distinct concept in the domain.

For the completeness analysis, we investigated how the different chosen experiments can be properly specified using the ExpDSLv1. The following aspects were assessed during the specification: Goals, Hypotheses, Subhypotheses, Design of experiment (DoE), Independent variables (controlled variable), Dependent Variables, Metrics, Measurement Instruments, Characterization/ Contextualization, Data Collection Procedure, Experimental roles, Statistical Analysis Technique, and Questionnaires. Our analysis considered three levels regarding the completeness of the specification: (i) supported– it was possible to specify the aspect; (ii) partially supported –the specification required adjustments to the modeling of the experiment aspect; and (iii) not supported – the DSL cannot specify that specific experiment aspect.

The expressiveness analysis involved verifying if each ExpDSLv1 construct was used to specify exactly one concept during the specification of the chosen experiments. Thus, if there are constructs being used to specify different abstractions of the experiments, our analysis consisted on the identification of such scenarios, which indicate a lack of expressiveness of the DSL.

III. ANALYSIS OF THE RESULTS

A. General Results

The modeling of controlled experiments in the study revealed that the investigated DSL successfully addressed most of the evaluated criteria. The completeness analysis showed that 60% of experimental aspects of the modeled experiments were supported, 15% partially supported, and 25% not supported. On the other hand, the expressiveness criteria revealed that there was only one ExpDSLv1 construct, the Metric element, which was used to specify three different concepts of the specified experiments.

Regarding the completeness of modeled experimental aspects – about 60% – were successfully satisfied. On the other hand, the results also show that a high percentage of experimental aspects was only partially supported (15%) or not supported (25%) by our approach. This demanded the investigation of how we could improve the ExpDSLv1 and our approach to allow their adequate specification. There were a few cases where the experiment aspect was not specified because it was not part of the experiment (n/a – not applicable).

Only three concepts were not supported for all modeled experiments, which are the dependent variable, measurement instruments, and the statistical analysis technique. These results motivated to propose ExpDSLv1 extensions to include: (i) a *DependentVariable* element that relates this concept with hypotheses and metrics, allowing the traceability among them and facilitating the conduction of meta-analysis in the future; and (ii) a statistical analysis element that allows documenting the statistical test which can be used for each experiment hypotheses.

There were also some scenarios where experimental aspects (such as metrics and experimental roles) have been only partially satisfied during the DSL completeness assessment. The metric element, for example, only allows specifying metrics regarding execution time for activity/task, and metrics that are quantified based on the produced artifacts. This result has also motivated the extension of ExpDSL to specify a new kind of metric related to any collected data informed by the user during an activity/task execution of the experiment process. Section III.C shows how we have improved the expressiveness of the language by extending the metric concept. It also discusses the lack of expressiveness of ExpDSLv1 to specify the design of experiment (DoE) and statistical analysis technique, and how language users can deal with that.

Next sections detail the results of the study regarding the modeling of different experimental aspects by presenting limitations of our approach and proposing new improvements.

B. Completeness Results and DSL Improvement.

This section describes the main results of our completeness study considering the different experimental aspects that were modeled in ExpDSLv1. For the more affected elements, we show and discuss the obtained results when modeling the experiments. In addition, we also describe how those results have been used to propose improvements and new extensions for the ExpDSL (ExpDSLv2).

1) Hypotheses

ExpDSLv1 allows the definition of statistical hypotheses for the experiment through the Hypothesis element. In our study, it was possible to model all hypotheses of the experiments using such language element. Fig. 1 – (B) shows the hypothesis specification for the “Testing” experiment.

To enable automatic analysis of hypotheses after that experiment execution, we propose a more formal hypotheses definition. This allows specifying an expression that defines the hypothesis relating the factor/levels with a dependent variable. Fig. 2 - (A) shows the new formalization of hypotheses after the language extension. Each textual hypothesis can be decomposed in one or more formal hypotheses, which associate the Factor or Treatment with the dependent variable.

There are two additional benefits provided by this DSL extension: (i) it contributes to the automation of the statistical hypotheses analysis; and (ii) it supports the analysis of variables used in different experiments (meta-analysis). Besides, it allows the hypotheses specification in a more formal way, associating dependents variables and treatments.

2) Design of Experiment

ExpDSLv1 supports three experimental DoE: (i) completely randomized design (CRD); (ii) randomized complete block design (RCBD); and (iii) Latin square (LS). For this reason, it was not possible to model the designs used in the Human Factors and Requirements experiments, because both experiments used a simple prospective ex post facto quasi-experimental design [14] [15]. The same issue happens to the PBR experiment, which adopts a factorial design in blocks of size 2. The wrong choice of one of the existing supported DoE of our DSL could cause a problem during the model transformation, because the distribution of treatments will not correspond to the real design of these experiments.

For allowing the modeling of experiments that do not follow any of the supported DoE in ExpDSLv2, we extend the language to include the “Other” option. This change has a direct impact in the experiment configuration. Without knowledge about the experimental design, transformations of models specified in ExpDSLv2 are not able to automatically randomize and allocate treatments to participants and experimental materials. Therefore, the experimenter has to configure this information manually using the execution environment using the ExpDSLv2. Only after this manual configuration, the execution environment instantiates the workflow correctly to start the experiment execution.

The choice of the “Other” DoE also makes difficult automatic validations of ExpDSLv2 specification based on statistical design rules and assumptions. Our DSL editor can check, for example, if the requirement of the Latin Square DoE is satisfied: existence of two variables to block. Therefore, the “Other” option increases language flexibility but limits specification validations.

3) Dependent Variable

In ExpDSLv1, we can define metrics related to dependent variables but there is no explicit ways to specify them. Because of that, we have extended the ExpDSLv1 to include the modeling of dependent variable that can be mapped to: (i) metrics that are associated to the execution time of specific activities or tasks of the experiments; or (ii) metrics that are associated to fields (of process activities/tasks) and that will be informed by the participants or researchers (number of defects found, etc.) during the experiment execution.

Fig 2 – (B) shows the dependent variable specified for the "Testing" experiment. There are two response variables in this experiment: the number of valid terminated CR (*NumberofCR*), and the time to execute the tests (TestExecutionTime). The *TestExecutionTime*, for example, has a description and is associated to the metrics *TimeExecution1*, *TimeExecution2*, *TimeExecution3*, and *TimeExecution4*. These metrics are defined in the experiment specification in the Metrics View.

4) *Metrics*

All metrics of the experiments were modeled by using one of the two kind of existing metrics in ExpDSLv1: those related to activities/tasks execution time and those associated to an artifact. Participants inform the value of a metric by using a text box provided in a web form related to the participant workflow. Another option is to define an *ArtifactMetric* element, forcing the participant to send a file whose content was only a value (text or number), for example. All these options represent workarounds that do not allow associating the informed metric value with the experiment hypotheses or other specific element. Fig. 1- (G) shows the definition of the metric Reported CRs used in the "Testing" experiment. In this experiment, the participant will be asked to upload a file whose content represents the collected CRs for each reported task. The specification shows the metric defined to the process related to the Specific Technique and Feature 1 (SpecificTecFeature1_Tests). The metric name is "ReportedCR1". The output artifacts related to the CR reporting are listed in the artifacts attribute of this metric (artifacts "CR SP1_F1_1", "CR SP1_F1_2, etc.).

In our study, the specification of different experiments using such strategy indicated the need to provide support to inform the metric value associated to an activity/task in the experiment. Such support contributes to not only explicitly identify the metric, but also associate it with the experiment hypotheses thus contributing to the analysis phase. Therefore, we extended the ExpDSLv1 creating a new Metric type named *DataMetric* that can be used to specify metrics associated to artifacts and that can be collected by the experiment subjects and/or researchers. This adaptation also provides an association between a metric and a text or number variable – the collectedData attribute. It has to be defined during the metric definition, and it is also used and collected in the correspondent activity/task of the experiment process.

Fig. 2 - (G) shows the specification of the "ReportedCR1" metric in ExpDSLv2. It has a description and a collectedData text attribute named "CR". This "CR" collectedData represents a data that will be gathered during the experiment execution, according to the process specification. Fig. 2 – (G) also illustrates that this "ReportedCR1" metric is associated to the "Number of CR" dependent variable. This will improve the traceability between dependent variable, metric and associated activity/tasks fields, facilitating their collection during the experiment execution and their usage during the experiment analysis.

5) *Data collection procedure*

In ExpDSLv1, data collection procedures are specified as a process in the Process View, containing activities, tasks, roles and artifacts. Our DSL does not currently support loops and conditionals paths but almost all the experiment could be modeled as sequential procedures. The semantic related to link the process to the correspondent treatment combination, Link element in ExpDSLv1 (Fig. 1 – E), was moved to the process definition (Fig. 2 – E), so the Link element was removed from the DSL. This element was not used in the quasi-experiment definitions (Human Factors and Requirements).

6) *Statistical Analysis Technique*

There was no support for defining analysis techniques in the experimental DSL. As the choice of the statistical test depends on the experiment design (DoE), we extended ExpDSLv1 to allow the selection of the statistical analysis technique from a list. Although this information is currently used only to document the experiment, our workflow engine is being extended to support the automatic application and analysis of the statistical tests of the experiments after the collection of the data of interest during the analysis phase. Fig. 2 – (D) shows the statistical analysis techniques chosen for the "CFT" experiment: McNemar to test hypothesis H1, and Wilconox to test hypotheses H2 e H3. The current list of possible techniques for testing hypotheses is: Chi-2, Binomial Test, t-test, F-test, McNemar Test, Mann-Whitney, Paired t-test, Wilcoxon, Sign test, ANOVA, Kruskal-Wallis and, for any other test, "Others". More than one type can be specified for each hypothesis.

C. *Expressiveness Results*

This section describes the analysis of the expressiveness criterion. We have mapped each ExpDSLv1 element/construction regarding the domain concept in order to evaluate if each language construction is associated to only one distinct concept in the domain.

For ExpDSLv1 we observed that the *ArtifactMetric* element was used to model metrics related to artifacts, questionnaires, and user data (data collected during the task/activity execution). This means that distinct concepts in the domain were expressed using the same abstraction of the DSL. In order to overcome this limitation, we created the *DataMetric* element in ExpDSLv2, which provides different ways to specify these three domain concepts: metric related to artifacts, questionnaires, and collected data. Fig. 2 - (G) shows a metric modeled as a *collectedData* element in ExpDSLv2. This *defectNumber* variable is referenced in each activity where the data have to be collected. It means that during the experiment process execution the user will be asked to inform this value during that specific activity.

We also observed an expressiveness limitation in ExpDSLv2 related to the language extension that defines the "Other" type to refer any other type of Design of Experiment (DoE) besides CRD, CRBD, and Latin Square. It has improved the completeness of the language (Section III.B), but on the other hand, it affects the expressiveness of the DSL, because this element can be used to express different concepts of the domain (factorial design, quasi-experimental designs, within subjects design, and so on). In order to improve the language coverage, we opted to accept this expressiveness decreasing because it allows the specification and guided execution of a higher number of experiments. The same problem of expressiveness happens to the AnalysisTechniqueType element of ExpDSLv2, which also has the option to the "Other" type.

IV. THREATS TO VALIDITY

One threat to the validity of this study is related to the choice of the experiments to be modeled. This choice defines for which types of experiments the conclusions are valid, restricting the extent to which the results can be generalized. We control this threat by selecting real experiments from different areas and with different DoE.

Another threat to validity of this study is called *Reliability* [1], which is concerned with to what extent the data and the analysis are dependent on the specific researchers. We control this threat by modeling and validating the experiments models with experimental software engineering researchers from Fraunhofer IESE and Universidad Politecnica de Madrid (UPM), which were not involved in the DSL development.

V. RELATED WORK

Whereas some existing research work explores the development of automated environments to support the conduction of controlled experiments [16], most of them do not describe in details how the specification of the different experimental aspects is addressed. Only a few research works have presented ways to formalize experiments in software engineering. Garcia et al. [10] present an ontology to describe experiments that have predicates to generate a plan for the experiment. Siy and Wu [11] present an ontology that allows defining an experiment and checking some constraints regarding validity threats to the experimental design, which are extracted from a set of experiments. Cartaxo et al. [9] present a graphic domain specific language to model experiments and generate a textual plan of it. However, all these research works do not provide an environment or tool support that allow interpreting and executing the controlled experiments based on their specifications. In addition, they have not conducted empirical studies to evaluate the completeness and expressivity of their specification mechanism – ontology or DSLs – through the modeling of different experiments, as we have presented in this paper.

VI. CONCLUSIONS AND FUTURE WORK

This paper presents an empirical study of evaluation and improvement of a DSL that supports the formalization of controlled experiments. In our study, several experiments reported by the software engineering community were specified using the DSL. These specifications were evaluated against the completeness and expressiveness criteria. The results of our study demonstrated the value of the approach, but on the other hand exposed a series of improvement opportunities. These improvements were discussed and addressed in a new version of the domain-specific language also evaluated in this paper.

As part of future work of this research, we are preparing two new studies to be conducted: (i) the first one is a survey that will be performed and applied to experts from the experimental software engineering community to collect feedback about the DSL; and (ii) the other one involves the usage of the improved version of the experimental DSL to conduct controlled experiments by other research groups in order to assessing the usability and performance of the approach. Finally, we also plan to extend the study presented in this paper to consider other existing criteria adopted in DSL evaluations [13].

ACKNOWLEDGMENTS

This study is supported by the program "Ciência sem Fronteiras", a joint initiative from Ministério da Ciência, Tecnologia e Inovação (MCTI) and Ministério da Educação (MEC) of Brazil, through CNPq and CAPES. It is partially supported by the National Institute of Science and Technology for Software Engineering (INES), funded by CNPq, grants 573964/2008-4 and 552645/2011-7, and by FAPERN, CETENE and CAPES/PROAP.

REFERENCES

1 Wohlin et al. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2012.

2 Juristo, Natalia and Moreno, Ana M. *Basics of Software Engineering Experimentation*. Kluwer Academic Publisher, Madrid, 2010.

3 Jedlitschka et al. Reporting Experiments in Software Engineering. In *Guide to Advanced Empirical Software Engineering*. Springer Science+Business Media, 2008.

4 Mendonça et al. A Framework for Software Engineering Experimental Replications. *IEEE ICECCS* (2008).

5 Hochstein et al. An Environment for Conducting Families of Software Engineering Experiments. *Advances in Computers*, 74 (2008), 175–200.

6 Sjøberg et al. Conducting Realistic Experiments in Software Engineering. *ISESE* (2002).

7 Travassos et al. An environment to support large scale experimentation in software engineering. *13th IEEE ICECCS* (2008), 193-202.

8 Freire et al. A Model-Driven Approach to Specifying and Monitoring Controlled Experiments in Software Engineering. In *PROFES*. LNCS, Paphos, Cyprus, 2013.

9 Cartaxo et al. ESEML: empirical software engineering modeling language. (New York 2012), Workshop on Domain-specific modeling.

10 Garcia et al. An Ontology for Controlled Experiments on Software Engineering. (San Francisco 2008), SEKE.

11 Siy, H. and Wu, Yan. An Ontology to Support Empirical Studies in Software Engineering. (Fullerton 2009), ICCEI.

12 Freire, M. A.. A Model-Driven Approach to Formalize and Support Controlled Experiments in Software Engineering. (Baltimore 2013), Proceedings of the ESEM Doctoral Symposium (ESEM 2013).

13 Kahraman, G. and Bilgen, S. A framework for qualitative assessment of domain-specific languages. *SoSym Journal* (November 23, 2013), 1-22.

14 Acuña et al. How do personality, team processes and task characteristics relate to job satisfaction and software quality? *IST*, 51, 3 (2009), 627–639.

15 Aranda et al. In Search of Requirements Analyst Characteristics that Influence Requirements Elicitation Effectiveness: a Quasi-experiment. (Madrid 2012), INTEAMSE.

16 Freire et al. Automated Support for Controlled Experiments in Software Engineering: A Systematic Review. In SEKE (Boston/USA 2013).

17 Accioly et al. Comparing Two Black-box Testing Strategies for Software Product Lines. *SBCARS* (2012).

18 Jung et al. A Controlled Experiment on Component Fault Trees. In *Conference on Computer Safety, Reliability and Security (SafeComp)* (Toulouse 2013).

19 Basili et al. The Empirical Investigation of Perspective-Based Reading. (1996).

20 Aleixo et al. Modeling Variabilities from Software Process Lines with Compositional and Annotative Techniques: A Quantitative Study. (Paphos 2013), PROFES.

21 Cirilo et al. Configuration Knowledge of Software Product Lines: A Comprehensibility Study. (Porto de Galinhas 2011), Workshop on Variability & composition.