Empir Software Eng (2017) 22:2206–2236
DOI 10.1007/s10664-016-9483-z

Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment

Arne N. Johanson[1] · **Wilhelm Hasselbring**[1]

Published online: 22 November 2016

Abstract It is a long-standing hypothesis that the concise and customized notation of a DSL improves the performance of developers when compared with a GPL. For non-technical domains—e.g., science—, this hypothesis lacks empirical evidence. Given this lack of empirical evidence, we evaluate a DSL for ecological modeling designed and implemented by us with regard to performance improvements of developers as compared to a GPL. We conduct an online survey with embedded controlled experiments among ecologists to assess the correctness and time spent of the participants when using a DSL for ecosystem simulation specifications compared with a GPL-based solution. We observe that (1) solving tasks with the DSL, the participants' correctness point score was —depending on the task— on average 61 % up to 63 % higher than with the GPL-based solution and their average time spent per task was reduced by 31 % up to 56 %; (2) the participants subjectively find it easier to work with the DSL, and (3) more than 90 % of the subjects are able to carry out basic maintenance tasks concerning the infrastructure of the DSL used in our treatment, which is based on another internal DSL embedded into Java. The tasks of our experiments are simplified and our web-based editor components do not offer full IDE-support. Our findings indicate that the development of further DSL for the specific needs of the ecological modeling community should be a worthwhile investment to increase its members' productivity and to enhance the reliability of their scientific results.

Keywords Domain-specific languages (DSLs) · Program comprehension · Computational science · Scientific software development

Communicated by: Romain Robbes

✉ Arne N. Johanson
arj@informatik.uni-kiel.de

Wilhelm Hasselbring
wha@informatik.uni-kiel.de

[1] Software Engineering Group, Kiel University, 24098, Kiel, Germany

1 Introduction

In science and research, we observe an increasing use of software. Often scientific experiments are conducted in virtual research environments with computer-based simulations, which are typically implemented by the scientists themselves. Traditionally, these scientific software developers employ no or only few software engineering methods and techniques while developing large simulation software systems. Therefore, specific software support is required to efficiently enable such scientific work.

Many software engineering approaches that address the resulting challenges of computational science employ DSLs (Fowler 2010; Stahl and Völter 2006; da Silva 2015; Mernik et al. 2005; Kosar et al. 2008). By using DSL, they intent to increase the productivity of the scientific software developers and to enhance the maintainability of their software solutions. Among those approaches are MDE4HPC (Palyart et al. 2012), the approach of suites of graphical DSL for science by Almorsy et al. (2013), and the Sprat Approach (Johanson et al. 2016, Johanson and Hasselbring 2014a; b).

Justification for relying on DSL instead of on GPL in this matter is provided by the long-standing hypothesis that:

> "Because of appropriate abstractions, notations and declarative formulations, a DSL program is more concise and readable than its GPL counterpart. Hence, development time is shortened and maintenance is improved." (Consel and Marlet 1998)

However, in a recent systematic mapping study on DSL, Kosar et al. (2016) find that there is a lack of evaluation research, in particular controlled experiments, to back this hypothesis.

In this paper, we contribute to the empirical study of DSL by evaluating the Sprat Ecosystem DSL (Johanson and Hasselbring 2014a), which is a DSL for specifying high-performance marine ecosystem simulation experiments, for its effectiveness and efficiency. For this purpose, we conducted an online survey including controlled experiments to compare the correctness and the time spent of experts from the domain of ecosystem simulation in solving typical ecosystem simulation specification tasks with our DSL and with a GPL-based solution. By measuring correctness point scores and time spent on the exercises, we evaluate the effectiveness and efficiency of the Sprat Ecosystem DSL. We focus on the domain of high-performance ecosystem simulation as it is representative of the HPC branch of computational science.

The main outcomes of our study are:

1. Solving tasks with the DSL, the participants' correctness point score was—depending on the task—61 % up to 63 % higher than with the GPL-based solution and their time spent on the tasks was reduced by 31 % up to 56 %.
2. The Ecosystem DSL receives higher user ratings than the GPL-based solution with regard to quality characteristics such as simplicity of use and maintainability of solutions.
3. Most users with no or very moderate Java skills are themselves able to carry out basic maintenance tasks concerning the Java-based infrastructure of the Sprat Ecosystem DSL itself. This is made possible by employing another internal DSL embedded into Java for configuring the Ecosystem DSL.

These findings provide empirical evidence that our DSL can indeed increase the productivity of scientific software developers, who often have no formal software engineering training.

For the structure of this paper, we follow the guidelines for reporting experimental results in software engineering from Jedlitschka et al. (2008). Thus, after introducing related work

in Section 2 and the Sprat Ecosystem DSL in Section 3, we discuss in Section 4 the design of our study and introduce the detailed hypotheses to be tested. Section 5 describes the survey results and the statistical procedures used in their analysis. The discussion of results in Section 6, which includes an analysis of threats to validity, is followed by concluding remarks in Section 7.

The full material used for the survey (including all task descriptions and questionnaires) can be found in the appendices of this paper. To facilitate the verifiability and reproducibility of our results, the anonymized raw data and analysis scripts of the survey are available online (Johanson 2016).

2 Related Work

As stated in the introduction of this paper, empirical research on the potential benefits of DSL with regard to user productivity is relatively scarce (Kosar et al. 2016).

Kieburtz et al. (1996) compared program generation from a textual DSL to code reuse mechanisms of a GPL (Ada templates) in the domain of message translation and validation for military command and control systems (see also Kieburtz and Hook, 1995). They hypothesized that the use of a DSL would increase productivity, reliability, flexibility, predictability (of development costs), and offer a better usability than a GPL. To test these hypotheses, Kieburtz et al. conducted a controlled experiment with four participants who held a BS or MS degree in computer science and had prior working experience with Ada. After the participants received training in both implementation technologies, they solved several tasks using both the DSL and the GPL (within-subject design) over the course of 12 weeks. As a result of the experiments, they found higher developer productivity and lower error rates for the DSL treatment. Like in our experiments, Kieburtz et al. used professionals as participants. Their participants, however, had a background in a technical domain (computer science), while our participants have a scientific background. Although their experiment lasts substantially longer for each participant than ours and includes professional training, our study includes nine times as many subjects.

Kosar et al. (2012) present results of a family of controlled experiments in which they compared how well computer science students can solve program comprehension tasks in technical domains using either a textual DSL or GPL. In total, they conducted three experiments covering the problem domains of feature diagrams, graph descriptions, and graphical user interface. In each experiment, the subjects participated in a short introduction to either the DSL or GPL they were supposed to use (between-subject design). Afterwards, the participants were given multiple tasks in which they had to analyze either DSL or GPL source code and had to answer a multiple-choice question regarding this code (related, e.g., to its effects or its maintenance). The authors report that—for the technical domains they analyzed—the subjects achieved higher accuracy and efficiency when using a DSL as compared with a GPL. The design and the analysis procedure of our experiments resemble those of the study by Kosar et al.. However, while they used undergraduate students and printed multiple-choice tasks on paper, we conducted our experiments among graduate students and domain experts using an online survey with open-ended questions (our subjects had to write code). Furthermore, instead of focusing on a technical domain, our study analyzes the potential benefits of DSL in ecology and therefore extends the findings of Kosar et al. to a scientific domain in which users typically have only moderate programming experience.

Hoisl et al. (2014) employed a controlled experiment to compare different notations for specifying scenario-based model tests (natural-language, diagrammatic, and fully-structured textual notation). Among the tested notations, the natural-language variant, which could be implemented as a DSL, leads to the highest productivity of the participants and is preferred by them as shown by the results of an ex-post questionnaire. Along similar lines, Meliá et al. (2015) studied the influence of graphical vs. textual DSL notations on the productivity and satisfaction of computer science students solving software maintenance tasks. In their experiment, the students preferred the textual notation and were also more efficient using this notation. However, note that the experiment of Meliá et al. did not include a GPL version of the tasks. Similarly, Ricca et al. (2010) report on a controlled experiment in which they assessed the comprehension of requirement specifications (which can be regarded as a DSL) with or without the presence of screen mockups (which can be understood as a graphical DSL notation). They find that the graphical notation improves the comprehensibility of functional requirements. As with the experiment of Meliá et al., in the experiment of Ricca et al., there is no direct comparison of the DSL notations with a GPL notation.

The related work mentioned so far focuses exclusively on the evaluation of DSL for technical domains—i.e., domains in which the developers are typically well-trained in programming and software engineering techniques. For non-technical domains—such as ecological modeling and scientific software development in general—we are not aware of any controlled experiments that would assess the impact of DSL on developers in such domains, who typically have no formal computer science education. However, Prabhu et al. (2011) in their survey of 114 researchers at Princeton University involved in computational science find that "DSL programmers report higher productivity and satisfaction" when compared with their colleagues who primarily use GPL.

3 The Sprat Marine Ecosystem DSL

The Sprat Ecosystem DSL is an external DSL that allows to specify ecosystem simulations (with a focus on marine systems) in a declarative way as shown in Fig. 1. Specifically, complex 3-D ecosystem models of the HPC community based on PDE are targeted (Johanson and Hasselbring 2014a).

A simulation description in the Sprat Ecosystem DSL consists of several top-level entities (Ecosystem, Output, Input, Species) that possess properties which describe the entity. Most of these properties have a constant numerical value given by an expression with a unit. The following code snippet gives an example of a top-level species entity containing a single attribute:

```
Species Herring {
   SwimmingSpeed: 8 [cm/s]
}
```

Unit support is vital for such a description language as there are numerous popular examples of mission-critical failures resulting from unit inconsistencies in numerical software (e.g., the Mars Climate Orbiter crash due to the mixed use of non-/metric units; Knight, 2002). If a unit is missing, the editor issues a warning and offers a quick fix that adds a unit of the correct quantity category to the expression (which would be kilograms in the case depicted in Fig. 1). Unit conversions (e.g., from degree Fahrenheit to degree Celsius) are automatically carried out by the DSL.

As some properties might be specified in relation to another quantity (e.g., a growth coefficient is specified for a certain temperature), a modifier can be introduced to these

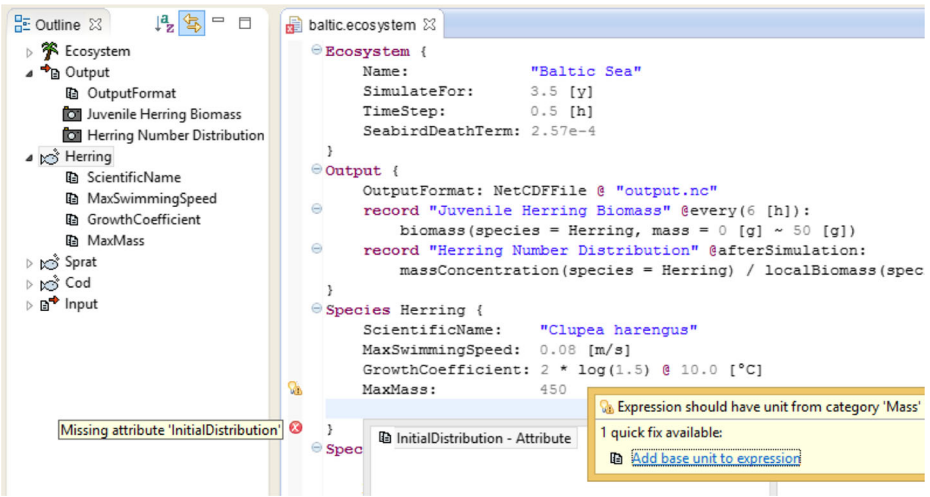


Fig. 1 Editor of the Sprat Ecosystem DSL featuring syntax coloring, high-level error messages, autocomplete, and quick fixes

properties with the `@` keyword:
`GrowthCoefficient: 0.1 @ 10 [°C]`,
which means "the growth coefficient is 0.1 at 10 degree Celsius."

Another keyword of the language is `record`. It can be used in the output entity to let the user describe which data should be collected during a simulation run via record expressions. This allows to aggregate the information already while the simulation is running and, thus, makes it unnecessary to store *all* the data generated by the simulation (which typically is a huge amount). Within record expressions, there are special functions to refer to model data. These functions are called using named parameters for better readability and the arguments can be intervals (`from ~ to`) with optional endpoints. The following code example demonstrates the use of record statements in the output entity:

```
Output {
   record ''Juvenile Herring Biomass'' @every(6 [h]):
     biomass(species = Herring, mass = 0 [g] ~ 50 [g])
}
```

As can be seen from the more detailed example in Fig. 1, the structure of the DSL is straightforward and only contains concepts that the target domain experts should be familiar with. Nonetheless, it is critical for the acceptance of any DSL to guide users while they construct models in the language. To this end, the editor of the DSL offers a list of content proposals at any given position in the document. Not only do these context-sensitive suggestions include isolated items, such as keywords, functions, and units, but also complete templates for, say, a new species entity and all its necessary properties. An example of this feature is displayed at the bottom of Fig. 1, where the name of the last missing species property (and only this missing one) is proposed. As long as not all necessary properties are specified, meaningful error messages are raised in appropriate locations. Beyond model completeness, the validator of the DSL checks various other constraints to ensure that the description of the simulation is sound and will result in a successful simulation run.

An overview on the meta-model of the Sprat Ecosystem DSL is presented in Fig. 9 on page 27 in the Appendix.

4 Study Design

In this section, we pose our research questions and describe the design of our study. The questions are answered by relying on data from both controlled experiments and user feedback.

4.1 Goals and Research Questions

We intend to investigate to which degree the Sprat Ecosystem DSL meets the needs of ecologists and how it compares with a GPL-based solution. We formalize this main research goal (RG) utilizing the goal definition template from the (GQM) method (Basili et al. 1994; van Solingen 1999) as follows:

RG: Analyze the Sprat Ecosystem DSL for the purpose of evaluation with respect to its suitability from the viewpoint of ecologists in the context of the implementation and maintenance of marine ecosystem simulations.

In order to derive an experiment procedure from this main goal, we explicate it by defining three RQ.

- **RQ1:** How accurately and how efficiently do ecologists carry out typical tasks in the implementation and maintenance of marine ecosystem simulations using the Sprat Ecosystem DSL compared to using a GPL?
- **RQ2:** How do ecologists judge the quality of the Sprat Ecosystem DSL compared to a GPL with regard to commonly accepted quality standards for programming languages and API for domain-specific tasks?
- **RQ3:** Are most users with no more than very moderate Java skills able to carry out basic maintenance tasks concerning the Ecosystem DSL infrastructure?

These research questions and the theoretical constructs involved are operationalized in Section 4.4, where we formulate our hypotheses to be tested. In this section, we also complete the GQM tree by listing the metrics we employ. Note that we use the GQM method only to achieve a clear top-down presentation and, therefore, adopt only selected aspects of the method.

4.2 Participants

According to our RG, we intend to analyze the Sprat Ecosystem DSL from the perspective of ecologists. In order to recruit suitable ecologists for our survey, we sampled web pages of research groups working in the field of (marine) ecology. From each research group we usually contacted a single person, requesting to forward our call for participation in the survey within their group. Additionally, we posted the link to the survey on a message board of the ICES.[1]

In total, we approached 16 research groups within a sampling period extending over about five weeks. The groups were located in Germany, Canada, the USA, and France (with descending frequency).

[1] http://www.ices.dk

This sampling strategy ensured that we would only contact domain practitioners with at least a graduate degree or being in the process of finishing such a degree (which we additionally verified with a questionnaire). In order to check whether the participants were actually working in the field of (marine) ecology, we collected the discipline of their highest academic qualification as well as their current occupation.

In total, 59 subjects took part in the survey, out of which 40 completed it (32 % drop-out rate). Our sampling design implies that we cannot give reasonable estimates of the answer rate. This is because we cannot be sure whether the persons we contacted in each research group actually forwarded the message to their colleagues.

The participants were informed about the purpose of the study and were guaranteed complete anonymity. They received no monetary compensation and participation was completely voluntary. The only incentive we gave was the reminder that their participation in the survey would support the development of DSL for their field.

4.3 Experimental Material and Tasks

In the following, we describe the data collection instruments and the tasks provided to the subjects. All materials used in the survey are included in the appendices of this paper.

Background Questionnaire The survey begins with a questionnaire that gathers information about the subjects' academic background and their prior experience with programming. The questions (13 in total) are a mix of open-ended questions (e.g., which programming languages they have experience with) and closed questions utilizing Likert scales (Likert 1932). Throughout the survey, we mostly use uneven scales with six possible answers to force choice between dis-/agreement. This helps to overcome the well-known response bias called "tendency towards the center" (Korman 1971).

Comprehension Tests The survey contains two tasks to test program comprehension with either the DSL or a GPL. Like the tests employed in the study of Kosar et al. (2012), our tasks focus on the implementation and maintenance phases of the software development life cycle as these phases are most demanding with regard to program comprehension activities (Hevner et al. 2005; Webb Collins et al. 2008). The tasks have been designed in collaboration with scientists who have worked with the Sprat Ecosystem DSL before (pilot users) so that their actual workflow is imitated when parametrizing and running high-performance ecosystem simulations with our DSL. In addition to imitating the scientists' workflow, the exercises are designed to cover all features of the Sprat Ecosystem DSL.

For each task, the subjects receive a short introduction with an example program that demonstrates how to solve a similar task with the respective programming language. To start an exercise, the participants have to click on a button after which they are shown a text editor component.[2] This component, in which the subjects enter their solution, initially contains the example program snippet given to them in the introduction to the task. The task description and the introductory text remain visible the whole time.

Below the text editor component, the participants find a small timer that displays a suggested maximum time to be spent on each exercise (four minutes for every task). The subjects can, however, choose to ignore this countdown and continue working after it

[2]For this purpose, we use CodeMirror (http://codemirror.net), for which we implemented a Sprat Ecosystem DSL syntax coloring plug-in.

expires. We included this counter to prevent participants who struggle with the tasks from dropping out of the survey due to frustration about how much time they have to spend on it.

In order to form a well-founded judgment about their experience with the DSL and the GPL, the participants have to be able to know whether their solutions are correct or not. Therefore, while solving the exercises, the subjects can, at any time, click on a button to check their current attempt at a solution for errors.

For the GPL in our survey, we chose C++ for two reasons. First, C++ is prevalent in the HPC branch of computational science and it is often employed for implementing modern high-performance simulation (libraries) (Prabhu et al. 2011; Basili et al. 2008; Faulk et al. 2009). Since the Sprat Ecosystem DSL is designed especially for high-performance ecosystem simulations, we intend to compare the DSL to a programming language ecologists would have to use to parametrize such a high-performance ecosystem simulation, which nowadays is likely to be C++. The second reason for choosing C++ is that the default code generator of the Sprat Ecosystem DSL produces C++ code, which we could adopt as a natural starting point for an alternative implementation for the control in our experiment.

The workflow of scientists using the Sprat Ecosystem DSL consists of two steps: first, they specify the scenario to be simulated (parametrization) and, second, which output they need from the simulation and how it is to be aggregated (data recording). Hence, the first task to be solved by the subjects is concerned with specifying parameters for a fictitious ecosystem simulation. The participants are given tables of parameters that consist of name/value pairs with measurement units and a short description of how to implement such a name/value pair with the DSL or the GPL, respectively (for the full task descriptions see Figs. 10 and 11 in the Appendix). When solving the C++ version of this task, the subjects have to perform simple unit conversions for some parameters (and are warned about this) as C++ does not feature the concept of units and their conversion. In Fig. 2, we show the example code given to the subjects and a possible solution for the parametrization task for both the DSL and the GPL treatment.

The second task deals with the recording and aggregation of data during a simulation run. The participants have to implement "recorders" for certain aggregated quantities (such as the total wet biomass of the system). For the GPL, we describe an API that allows to define such recorders (for the full task descriptions see Figs. 12 and 13 in the Appendix). In Fig. 3, we show the example code given to the subjects and a possible solution for the data recording task for both the DSL and the GPL treatment.

For each exercise we measure *correctness* in percent and *time spent* (for a detailed description how correctness is measured, see Section 5.1). Here, time spent means the time from starting the exercise (by clicking the button) to submitting the solution *while the web page with the editor component is active/in focus*. With the latter restriction, we correct for some of the possible interruptions of the subjects (such as receiving and reading an e-mail while solving the task).

Comprehension Feedback Questionnaire After completing both tasks with both the DSL and the GPL, the subjects are asked to rate the programming languages with respect to certain quality characteristics:

1. Level of abstraction
2. Simplicity of use
3. Ease of comprehension

```
Ecosystem {
  OxygenLevel: 0.1 [mmol/l]
  WindSpeed: 12 [km/h]
}
Species cod {
  MaxMass: 1 [kg]
  IngestionExponent: 0.5
}
Species herring {
  MaxMass: 100 [g]
  IngestionExponent: 0.75
}
```

(a) DSL Example

```
Ecosystem {
  Area: 1000 [m^2]
  PredatorPreyRatio: 3.5
  SeabirdPredationFactor: 0.025 [d^-1]
}
Species cod {
  MaxLength: 1.2 [m]
  MaxSwimmingSpeed: 2.52 [km/h]
  GrowthRate: 0.1 [d^-1]
}
Species herring {
  MaxLength: 45 [cm]
  MaxSwimmingSpeed: 40 [cm/s]
  GrowthRate: 0.15 [d^-1]
}
```

(b) DSL Solution

```
struct ModelParameters {
  static constexpr double OxygenLevel = 0.1;
  static constexpr double WindSpeed = 12.0;
  const real MaxMass[2];
  const real IngestionExponent[2];
  ModelParameters() :
    MaxMass {1.0, 0.1},
    IngestionExponent {0.5, 0.75}
  {}
};
```

(c) GPL Example

```
struct ModelParameters {
  static constexpr double Area = 1000;
  static constexpr double
              PredatorPreyRatio = 3.5;
  static constexpr double
              SeabirdPredationFactor = 0.025;
  const real MaxLength[2];
  const real MaxSwimmingSpeed[2];
  const real GrowthRate[2];
  ModelParameters() :
    MaxLength {1.2, 0.45},
    MaxSwimmingSpeed {0.7, 0.4},
    GrowthRate {0.1, 0.15}
  {}
};
```

(d) GPL Solution

Fig. 2 Example code and a possible solution for both the DSL and the GPL version of the parametrization task

4. Absence of technicalities from the syntax
5. Maintainability of solutions

These categories are derived from commonly accepted quality attributes for programming languages and API for domain-specific tasks (Kolovos et al. 2006). Again, the questionnaire uses forced-choice six-point Likert scales to measure the responses.

In addition to the closed questions discussed above, the feedback questionnaire contains two open-ended questions allowing the subjects to elaborate on difficulties with the Sprat Ecosystem DSL and suggestions for improving it.

Infrastructure Maintenance Test In this last test, the participants are supposed to carry out the relatively simple maintenance task of adding a new attribute for fish species to the Sprat Ecosystem DSL (the name of the attribute is supposed to be `WetMassAtFecundity` and its unit type should be *mass*; for the full task description see Fig. 14). Note that in this task, the users are not confronted with code written in the Ecosystem DSL but rather have to make changes to the code that implements our DSL. For this purpose, they are supplied with the central configuration file of the Ecosystem DSL, which is a Java source code file with about 130 LOC. This Java code intentionally does not adhere to standard object oriented design principles. In contrast, it employs a style that is

```
Output {
  OutputFormat: NetCDFFile @ "output.nc"

  record "Cod wet biomass" @every(6 [h]):
    wetBiomass(species = cod)

  record "Her. offspring" @beforeSimulation:
    matureWetBiomass(species = herring) *
      fecundity(species = herring)
}
```

(a) DSL Example

```
Output {
  OutputFormat: NetCDFFile @ "output.nc"

  record "Herring dry biomass" @every(12 [h]):
    dryBiomass(species = herring)

  record "Total wet biomass" @afterSimulation:
    wetBiomass(species = cod) +
      wetBiomass(species = herring)
}
```

(b) DSL Solution

```
struct RecorderSetup {
  // Recorder: Cod wet biomass
  class RecImpl0 : public Recorder {
  public:
    using Recorder::Recorder;
  protected:
    real recordValue(DoFT * dof) {
      return wetBiomass(dof, 0);
    }
    RecordWhen when() {
      return RecordWhen::EVERY;
    }
    real interval() {
      return 2.16E4; // in seconds
    }
  };

  // Recorder: Herring offspring
  class RecImpl1 : public Recorder {
  public:
    using Recorder::Recorder;
  protected:
    real recordValue(DoFT * dof) {
      return matureWetBiomass(dof, 1) *
             fecundity(dof, 1);
    }
    RecordWhen when() {
      return RecordWhen::BEFORESIMULATION;
    }
  };
};
```

(c) GPL Example

```
struct RecorderSetup {
  // Recorder: Herring dry biomass
  class RecImpl0 : public Recorder {
  public:
    using Recorder::Recorder;
  protected:
    real recordValue(DoFT * dof) {
      return dryBiomass(dof, 1);
    }
    RecordWhen when() {
      return RecordWhen::EVERY;
    }
    real interval() {
      return 2*2.16E4; // in seconds
    }
  };

  // Recorder: Total wet biomass
  class RecImpl1 : public Recorder {
  public:
    using Recorder::Recorder;
  protected:
    real recordValue(DoFT * dof) {
      return wetBiomass(dof, 0) +
             wetBiomass(dof, 1);
    }
    RecordWhen when() {
      return RecordWhen::AFTERSIMULATION;
    }
  };
};
```

(d) GPL Solution

Fig. 3 Example code and a possible solution for both the DSL and the GPL version of the data recording task

similar to forming a small embedded DSL that targets programming novices who are not familiar with Java. To solve the task, the subjects have to scan through the code to find a line that look like:

```
SPECIES_ATTRIBUTES.add(new SpratAttribute("MaxMass", MASS));
```

They have to copy this line and modify it so that it reads:

```
SPECIES_ATTRIBUTES.add(new SpratAttribute("WetMassAtFecundity",
                                          MASS));
```

The structure of the test page is the same as the one of the comprehension tests described above.

Maintenance Feedback Questionnaire This feedback questionnaire is concerned with assessing how the participants perceive the simplicity of the Java code during the test described above. Additionally, it measures the level of Java expertise.

4.4 Variables and Hypotheses

To answer the research questions **RQ1** and **RQ2**, we have to manipulate whether the subjects use the DSL or the GPL for the tests and questionnaires mentioned above while ensuring that the participants fit the intended user profile of the Sprat Ecosystem DSL. Additionally, we control for the experience with programming in general and with certain programming languages in particular. For **RQ3** to be answered, we have to control for the users' Java expertise.

We therefore introduce the following in-/dependent variables:

- Independent variables (controlled):
  - Academic discipline
  - Academic degree
  - Academic occupation
  - Programming experience in general
  - C++ experience
  - Java experience
- Independent variables (manipulated):
  - Programming language (DSL/GPL treatment)—within-subjects variable
  - Task (Parametrization/Recording)—within-subjects variable
  - DSL/GPL order
- Dependent variables:
  - Correctness point score of program comprehension tasks
  - Time spent on program comprehension tasks
  - Quality attributes: level of abstraction, simplicity of use, ease of comprehension, absence of technicalities, maintainability of solutions (see above)
  - Correctness of DSL infrastructure extension

To allow for statistical hypothesis testing, we formalize **RQ1-3** into the following set of four directed hypothesis pairs (for the complete GQM tree, see Table 1):

Correctness

H1$_0$: The mean correctness for both program comprehension tasks is less or equal for the DSL than for the GPL treatment.

Table 1 GQM tree for the evaluation of the Sprat Ecosystem DSL

Goal	Question	Metric
RG	RQ1	Correctness in percent (**H1**$_1$)
		Time spent (**H2**$_1$)
	RQ2	Rating score (**H3**$_1$)
	RQ3	Correctness (**H4**$_1$)

H1$_1$: The mean correctness for both program comprehension tasks is greater for the DSL than for the GPL treatment.

Time spent

H2$_0$: The mean time spent for both program comprehension tasks is greater or equal for the DSL than for the GPL treatment.

H2$_1$: The mean time spent for both program comprehension tasks is less for the DSL than for the GPL treatment.

Quality

H3$_0$: There exists a quality attribute for which the DSL mean rating is less than or equal to the GPL mean rating.

H3$_1$: For all quality attributes the DSL mean rating is greater than the GPL mean rating.

Maintainability by non-professionals

H4$_0$: Less than 90 % of the users with Java skills less than or equal to "beginner" are able to carry out basic maintenance tasks of the Sprat Ecosystem DSL.

H4$_1$: At least 90 % of the users with Java skills less than or equal to "beginner" are able to carry out basic maintenance tasks of the Sprat Ecosystem DSL.

Our operationalization of "most users" (**RQ3**) for **H4** as 90 % is, of course, contingent. However, we believe this to be in accordance with common sense.

The reason for choosing directed hypotheses for **H1**-**H3** is motivated by the rationale expressed throughout the literature that the abstract, domain-specific notation of a DSL should be easier to work with than a GPL. Therefore, for a programmer who is familiar with the concepts of the respective domain, program comprehension should be higher with a DSL than with a GPL (cf. Kosar et al, 2012).

4.5 Experimental Design

Our main experiment regarding program comprehension and quality (**H1**-**H3**) features a two-factor (DSL/GPL treatment and parametrization/recording task) within-subject design. The main reason for choosing a within-subject over a between-subject design is that in this way, each participant acts as their own control with regard to programming experience and domain expertise. This advantage, however, entails the drawback of possible intra-subject learning effects. Since we give each participant the same tasks to be solved with both the DSL and the GPL, it is likely that they acquire language-independent knowledge about the task that allows them to solve it better on their second encounter with it. This potential bias can be mitigated effectively by randomizing the order of treatments (Jones and Kenward 2014). Thus, we ordered the programming languages for the tasks randomly for each participant.

The DSL infrastructure maintenance experiment to test **H4** does not employ a standard modern experiment design. Thus, we use the term "exploratory" experiment. Although this experiment does not have an independent variable that we manipulate, it follows a clear experimental set-up and allows for valid hypothesis testing. With this design we will, of course, not be able to compare our DSL extension interface implementation to any other method nor establish a sound cause-effect relationship between our implementation approach and the participants' ability to carry out simple maintenance tasks. The experiment is meant to be a first step to explore whether experts from non-technical domains are able to extend DSL developed for them.

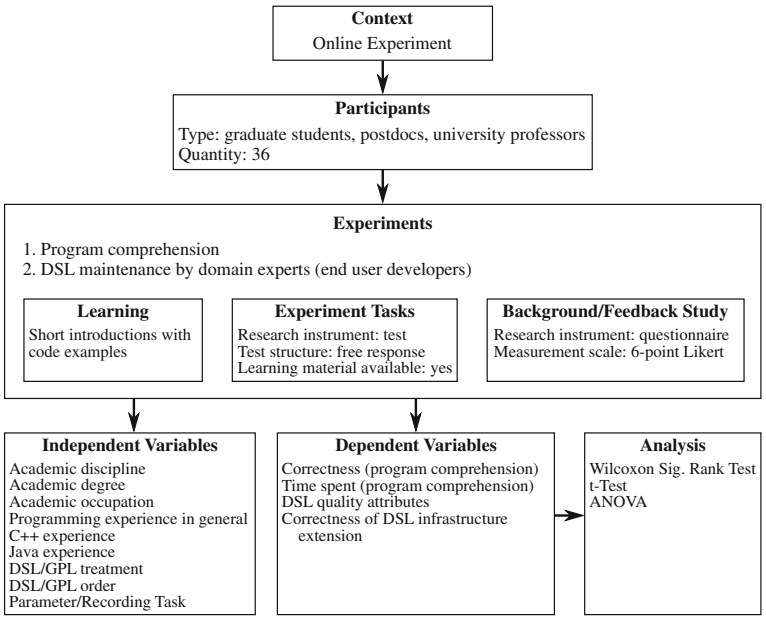


Fig. 4 Overview on experimental setup

4.6 Procedure

The data collection phase of our online experiment lasted for about five weeks and was conducted in January and February 2015. The order in which the data collection instruments were presented during the experiment corresponds to the order in which they are discussed in Section 4.3.

The survey does *not* include any dedicated learning phases for the subjects to familiarize with the programming languages to be used to solve the tasks. Instead, each task is preceded by a concise description of how to employ the corresponding language for the given task. These descriptions heavily rely on code examples in order to account for the fact that the participants potentially only have relatively moderate programming skills. Additionally, the very short introductions allowed us to make sure that their informational content is equal for both the DSL and the GPL.

Table 2 Academic disciplines of the participants

Discipline	Number	Fraction (%)
(Marine) Biology	14	38.9
(Marine) Ecology	8	22.2
Fisheries Science	4	11.1
Biological Oceanography	4	11.1
Marine Environmental Sciences	3	8.3
(Applied) Mathematics	3	8.3

Table 3 Occupations of the participants

Occupation	Number	Fraction (%)
Ph.D. Student	24	66.7
Postdoctoral Researcher	8	22.2
Master Student	3	8.3
Professor	1	2.8

Because of using directed hypotheses for the comparison of the DSL and the GPL, we do not have to take measures to counteract different levels of expertise with the languages: it is highly unlikely that any of the participants have previously been exposed to the Sprat Ecosystem DSL; if they have prior knowledge in C++, this would only increase the cogency of our alternative hypotheses.

A summary of the whole experimental setup is depicted in Fig. 4.

5 Analysis and Results

This section discusses the analysis procedure for the survey results and reports on descriptive and inferential statistics. As the threshold for statistically significant results we used $\alpha = 0.05$. All analyses were performed using *R 3.1.1*[3] and our R analysis scripts as well as the anonymized raw data can be obtained online (Johanson 2016).

In total, there were 59 participants, out of which 40 subjects completed the entire survey. This corresponds to a drop-out rate of approximately 32 %. We discarded all incomplete responses and further filtered the data set by applying the following two criteria:

1. *Click-through:* we discarded result sets from subjects spending less than 30 seconds on any task of the experiment as those were assumed to not really have tried to solve the task. Number of discards: 3
2. *Wrong discipline:* we discarded the answers from participants whose academic discipline was not related to ecological modeling. Number of discards: 1

This filtering resulted in $n = 36$ result sets that were further analyzed. About 52.8 % of these participants were randomly selected to solve each task first using the DSL.

The academic disciplines and current occupations of the subjects are summarized in Tables 2 and 3. Most (61.1 %) reported to be working in (marine) biology or, more specifically, (marine) ecology. The remaining participants pursue careers either in a multidisciplinary subject which involves ecology such as fisheries science or they approach ecological modeling with a background in mathematics.

As can be seen from Fig. 5, most participants have medium-level experience with programming—though there are two subjects who state that they have never written a program before. Average experience is lower with C++ and Java. The most-named programming languages that the users have used so far are: R (named 25 times), Matlab (19), C or C++ (12), Python (9), and Fortran (8). In this context it is noteworthy, that the interest in programming among the subjects is on average much higher than their expertise in this matter.

[3] http://www.r-project.org

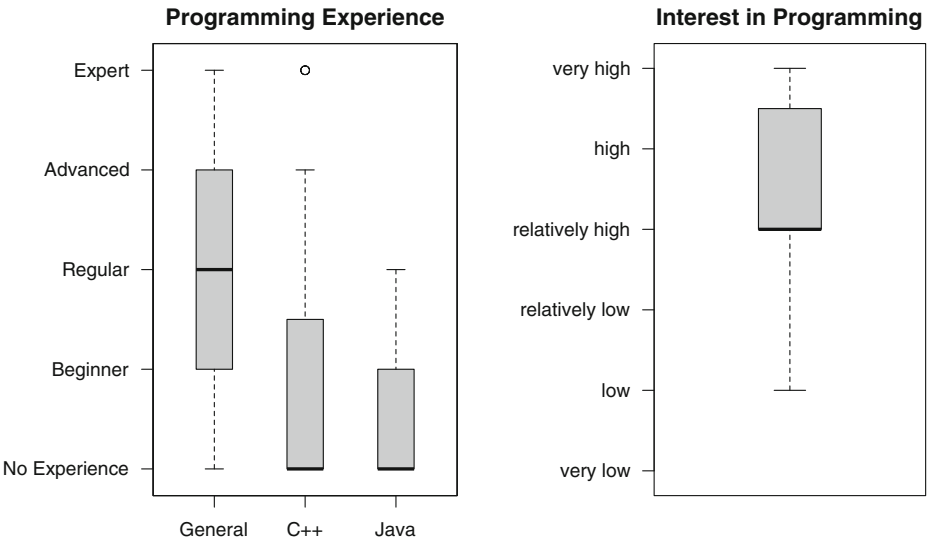


Fig. 5 Programming experience and interest in programming of the participants. Experience levels: **1.** No experience (never written a program, difficulty with reading), **2.** Beginner (problems writing easy programs, can read), **3.** Regular (can write basic programs, no problems reading), **4.** Advanced (can write complex programs), **5.** Expert (years of experience)

The following sections are concerned with testing the hypotheses **H1**-**H4** statistically. For this purpose, we report the results of the paired difference t-test, of Wilcoxon's signed rank test (Wilcoxon 1945), and of the Shapiro-Wilk test (Shapiro et al. 1968) for each task in isolation. Furthermore, for the program comprehension experiment, we present the results of a two-way ANOVA for repeated measures to test for interaction effects of our two factors *programming language* and *task*. Note that we perform one-tailed tests (for t-test and Wilcoxon) because our hypotheses are directed. Also note that we can assume homogeneous variances for the t-test because of our paired study design.

It is common practice to employ a "normality test"[4] like Shapiro-Wilk to decide whether to use a parametric (t-test) or a non-parametric (Wilcoxon) test. One could argue, however, that the central limit theorem ensures that for samples large enough (typically $n > 30$ is assumed), the mean of the samples is guaranteed to be approximately normal (Bortz and Schuster 2010). This means that in our case the assumptions for the t-test would always be met. Fortunately, we do not have to be concerned with this debate, as we will see that both tests always agree.

Whether or not an observed difference is significant does not say anything about its practical relevance: even the tiniest differences can become significant if the sample size is large enough. Therefore, for each significant difference in means μ_X and μ_Y between samples X_i and Y_i, we report the *effect size*

$$\delta = \frac{|\mu_X - \mu_Y|}{\sigma_D}, \tag{1}$$

[4]These tests are all formulated to test *against* normality (the alternative hypothesis is that the sample is *not* normal). This means that the test, strictly speaking, can never show that a sample is likely to be normal because, as Bortz and Döring (2006) put it, "a non-significant result says nothing."

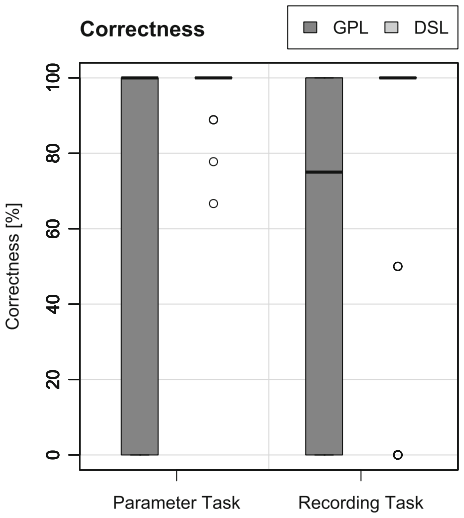


Fig. 6 Correctness of solutions for the parametrization and the recording tasks. The box plots for the DSL are degenerated because of more than 75 % fully correct solutions

where σ_D is the standard deviation of the sample difference $D_i = X_i - Y_i$. Defined in this way, the effect size δ corrects the absolute difference of the means for the variance that is present in the sample. $\delta \geq 0.2$ corresponds to *small*, $\delta \geq 0.5$ to *medium*, and $\delta \geq 0.8$ to *large* effects (Bortz and Döring 2006).

5.1 Correctness and Time Spent

To measure correctness, for each task we identified a number of program features (i.e., ecosystem parameters and recording specifications) that need to be implemented for a solution to be correct. Thus, correctness $C^{\kappa}_{i,j}$ for subject i and task j ($1 =$ Parameters, $2 =$ Recording) of treatment κ ($1 =$ GPL, $2 =$ DSL) is given as the quotient of the

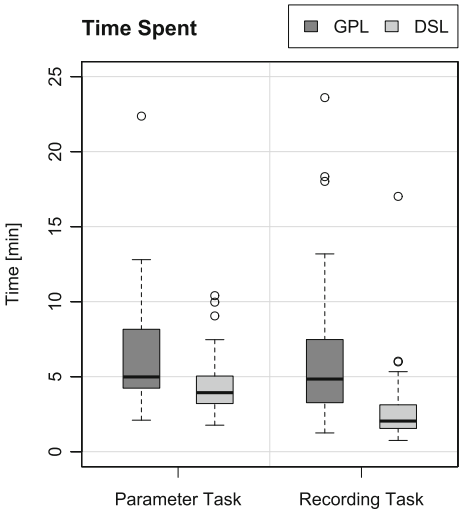


Fig. 7 Time spent on solving the parametrization and the recording tasks

Table 4 Comparison of correctness for comprehension tasks. Significant p-values printed in bold

	Correctness [%]			
	Parameters		Recording	
	GPL	DSL	GPL	DSL
Mean	60.8	97.8	52.8	86.1
Difference		+60.9 %		+63.2 %
SD	47.3	6.9	45.0	33.0
Median	100.0	100.0	75.0	100.0
Shapiro-Wilk p		$\mathbf{5.7 \cdot 10^{-7}}$		$\mathbf{6.2 \cdot 10^{-5}}$
t-Test p		$\mathbf{1.2 \cdot 10^{-5}}$		$\mathbf{2.0 \cdot 10^{-4}}$
Wilcoxon p		$\mathbf{1.0 \cdot 10^{-4}}$		$\mathbf{5.5 \cdot 10^{-4}}$
Effect Size δ		0.81 (large)		0.65 (medium)

number of correctly implemented features $\eta^{\kappa}_{i,j}$ divided by the total amount of features to be implemented v_j:

$$C^{\kappa}_{i,j} = \frac{\eta^{\kappa}_{i,j}}{v_j} \tag{2}$$

If syntax errors prohibit a solution from compiling, we set $\eta^{\kappa}_{i,j} = 0$.

Figures 6 and 7 give an overview of the participants' performance with regard to correctness and spent. For each task, more than 75 % of the subjects were able to solve it without errors using the Sprat Ecosystem DSL, which results in degenerated box plots for these cases in the correctness plot. Furthermore, no participant completely failed the parameter specification task with the DSL while there are 0 % correct solutions for both tasks with the GPL. The median efficiency for the DSL is greater than the median of the GPL among all categories. The largest difference in time spent can be observed for the recording task.

Descriptive and inferential statistics for the program comprehension tasks are summarized in Tables 4 and 5. It can be seen that the DSL solutions are on average about 61 % and 63 % more correct than those implemented with the GPL. Furthermore, with the DSL, participants implement solutions on average about 31 % and 56 % faster than with the GPL. The Shapiro-Wilk test succeeds for all categories (implying that the sample is most likely

Table 5 Comparison of time spent for comprehension tasks. Significant p-values printed in bold

	Time Spent [min]			
	Parameters		Recording	
	GPL	DSL	GPL	DSL
Mean	6.5	4.5	6.4	2.8
Difference		−30.7 %		−56.1 %
SD	4.0	2.1	5.0	2.8
Median	5.0	3.9	4.8	2.0
Shapiro-Wilk p		$\mathbf{1.7 \cdot 10^{-2}}$		$\mathbf{4.6 \cdot 10^{-4}}$
t-Test p		$\mathbf{2.2 \cdot 10^{-3}}$		$\mathbf{2.5 \cdot 10^{-4}}$
Wilcoxon p		$\mathbf{7.7 \cdot 10^{-4}}$		$\mathbf{6.6 \cdot 10^{-6}}$
Effect Size δ		0.51 (medium)		0.64 (medium)

not drawn from a normal distribution). As Wilcoxon's signed rank test and the t-test indicate a significant difference in means in the direction of our alternative hypotheses $\mathbf{H1}_1$ and $\mathbf{H2}_1$, we reject $\mathbf{H1}_0$ and $\mathbf{H2}_0$. The effect size δ is "large" for the correctness of the parametrization task and "medium" otherwise.

Results from the ANOVA of the correctness and time spent data are displayed in Table 6. The main effect of the factor *language* is statistically significant for both correctness and time spent. Both the main effect of the factor *task* and the interaction effect of the two factors are not statistically significant for correctness and time spent.

5.2 User Perceptions

The participants' perceptions of the DSL and the GPL with regard to the five quality categories explained in Section 4.3 are visualized in Fig. 8. The medians of the quality ratings are all higher for the DSL relative to the GPL. Also considered independently, the absolute ratings of the Sprat Ecosystem DSL speak in its favor: the medians of the DSL are "high" for all categories and not even a single outlier is lower than "relatively low" with respect to any quality attribute.

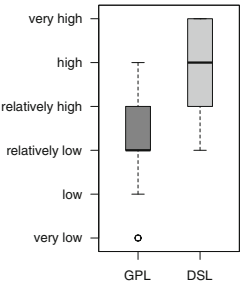
These observations are reinforced by the results of the statistical analysis given in Table 7. Both Wilcoxon's signed rank test and the t-test indicate a significant difference in the subjects' ratings in favor of the DSL. Therefore, we reject the null hypothesis $\mathbf{H3}_0$. For all categories we find a "large" effect size.

5.3 DSL Maintenance by the Users

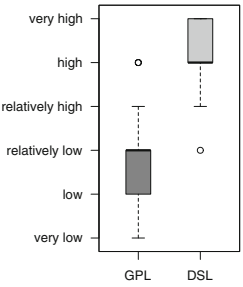
To test the hypothesis pair $\mathbf{H4}_0/\mathbf{H4}_1$ about the maintainability of the Sprat Ecosystem DSL infrastructure by non-professionals, we focus only on participants with Java skills less than or equal to "beginner". Among our sample with $n = 36$, there are 33 subjects that match this criterion. Of these 33, only one participant was *not* able to extend the DSL correctly with an additional attribute, which corresponds to a mean success rate of $\mu = 97.0\ \%$. To determine whether μ is significantly greater than or equal to 90 %, we perform a one sample, one-tailed t-test. This test results in a significant p-value of 0.0141, which leads us to reject the null hypothesis $\mathbf{H4}_0$.

Table 6 Results from the two-way ANOVA for repeated measures. Significant p-values printed in bold

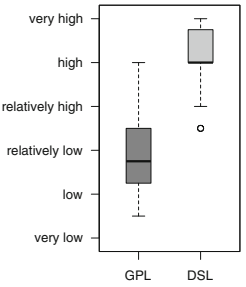
Factor	Df	Sum Sq.	Mean Sq.	F value	p value
Correctness					
Language	1	17791	17791	12.910	$\mathbf{4.6 \cdot 10^{-4}}$
Task	1	442	442	0.321	$5.7 \cdot 10^{-1}$
Language:Task	1	122	122	0.089	$7.7 \cdot 10^{-1}$
Residuals	136	187411	1378		
Time Spent					
Language	1	69.7	69.66	5.179	$\mathbf{2.4 \cdot 10^{-2}}$
Task	1	19.8	19.77	1.470	$2.3 \cdot 10^{-1}$
Language:Task	1	3.8	3.82	0.284	$5.9 \cdot 10^{-1}$
Residuals	136	1829.4	13.45		



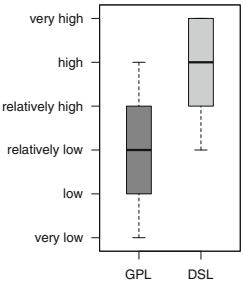
(a) Level of abstraction



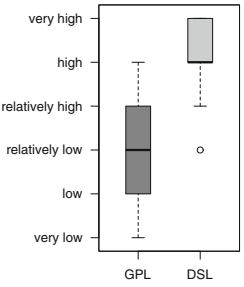
(b) Simplicity of use



(c) Ease of comprehension



(d) Absence of technicalities



(e) Maintainability of solutions

Fig. 8 Perceived quality of the GPL and the DSL

6 Discussion

In the following, we discuss the results of the statistical analysis presented above and examine possible threats to validity.

Table 7 Comparison of the DSL. Significant p-values printed in bold

	Rating [1 ("very low") to 6 ("very high")]									
	Abstraction		Simplicity		Comprehension		Abs. of technic.		Maintainability	
	GPL	DSL	GPL	DSL	GPL	DSL	GPL	DSL	GPL	DSL
Mean	3.28	4.97	2.69	5.03	2.83	5.15	3.06	4.97	3.17	5.06
SD	1.06	0.84	1.06	0.77	0.95	0.68	1.17	0.81	1.21	0.79
Median	3.00	5.00	3.00	5.00	2.75	5.00	3.00	5.00	3.00	5.00
Shapiro-Wilk p		$\mathbf{1.9 \cdot 10^{-3}}$		$\mathbf{4.8 \cdot 10^{-2}}$		$\mathbf{1.3 \cdot 10^{-1}}$		$\mathbf{3.3 \cdot 10^{-3}}$		$\mathbf{3.7 \cdot 10^{-2}}$
t-Test p		$\mathbf{1.4 \cdot 10^{-10}}$		$\mathbf{1.7 \cdot 10^{-12}}$		$\mathbf{4.9 \cdot 10^{-15}}$		$\mathbf{4.0 \cdot 10^{-10}}$		$\mathbf{3.2 \cdot 10^{-10}}$
Wilcoxon p		$\mathbf{2.8 \cdot 10^{-7}}$		$\mathbf{1.5 \cdot 10^{-7}}$		$\mathbf{1.2 \cdot 10^{-7}}$		$\mathbf{1.1 \cdot 10^{-6}}$		$\mathbf{3.9 \cdot 10^{-7}}$
Effect Size δ		1.45 (large)		1.73 (large)		2.13 (large)		1.39 (large)		1.40 (large)

6.1 Correctness and Time Spent

If you compare the effect size for correctness between the recording task and the parametrization task, you find that it is much lower for the former than for the latter (0.65 ("medium") vs. 0.81 ("large")). This means that the DSL is less effective in reducing complexity for the programmer for the recording task than it is for the parametrization task. This is surprising because one could argue that the Sprat Ecosystem DSL features more powerful domain-specific abstractions for the recording task (recorders, recording expressions, recording intervals) than for the parametrization task (here, the GPL lacks only the concept of units). Hence, using the DSL compared to the GPL should have a larger impact on the recording than on the parametrization task. The fact that we find the exact opposite could be explained by the observation that while a DSL can simplify the description of a solution, it cannot simplify the solution itself: the task of formalizing data aggregation is complicated for domain experts irrespective of which language they use to express this formalization. Or in other words: a DSL can support domain experts in expressing solutions but it cannot create the solutions for them.

This interpretation would also explain that, while the difference in mean time spent between DSL and GPL is much larger for the recording than for the parametrization task, the effect sizes are approximately equal (both are "medium"). Those subjects who were good at solving the tasks on the domain level could express their solution much more efficiently with the Sprat Ecosystem DSL than with C++. Those subjects who already struggled with the task on the domain level were inefficient in solving it, no matter in what language they expressed their solution (resulting in a large sample variance and thus a reduced δ-value).

Our comprehension experiment provides evidence that our declarative DSL for high-performance ecosystem simulation increases program comprehension when compared with the imperative/object-oriented language C++, which is ubiquitous in the HPC community. The results show that our DSL can be an effective tool to empower marine ecologists to write well-maintainable programs (see discussion of user perceptions below) by themselves without the need for extensive training in software engineering methods. The medium to large effect sizes indicate that these findings are relevant for practice and they imply that the development costs associated with our DSL should pay off relatively quickly.

6.2 User Perceptions

The ratings of the Sprat Ecosystem DSL indicate that the domain-specific abstractions of the language are well-implemented (Fig. 8a) and can be used with an accessible, concise syntax (Fig. 8b) without introducing unwanted technicalities (Fig. 8d). This implies that the domain meta-model of the DSL successfully captures the essential concepts of the ecosystem simulation domain and that the DSL employs a syntax that matches the jargon of the discipline better than it could be achieved with the GPL-based solution.

The higher rating of the DSL with regard to ease of program comprehension (Fig. 8c) and maintainability of solutions (Fig. 8e) shows that the Sprat Ecosystem DSL is more suitable to implement well-readable and well-maintainable software than the GPL.

6.3 DSL Maintenance by the Users

DSL are no exception to the rule that for most non-trivial software projects, maintenance accounts for a large amount of effort in the software life cycle (April and Abran 2012).

As the Sprat Ecosystem DSL is supposed to enable scientists to implement software by themselves, it appears to be desirable to also enable them to carry out basic maintenance of the DSL infrastructure themselves. The results of the infrastructure maintenance test indicate that the use of another embedded DSL for crucial parts in the implementation of the main DSL can be an effective means to achieve that.

6.4 Threats to Validity

This section discusses limitations of our study design that could potentially affect the validity of our findings. We distinguish between threats to internal and to external validity (Jedlitschka et al. 2008; Siegmund et al. 2015):

Threats to internal validity To what degree are the independent variables we measured or manipulated responsible for the effects observed in the dependent variables? Could instead unobserved variables be responsible for the effects?

Threats to external validity To what extent can our findings be generalized to other settings and populations (esp. to real world scenarios)?

In the following, we list only those threats to validity that are specifically relevant to our study. For threats to validity that apply to the results of experiments in general and to those in software engineering in particular, see Wohlin et al. (2012).

Internal Validity

Selection Unevenly distributed general performance levels of the subjects between the DSL and the GPL treatment group could influence the results. We circumvented this risk by choosing a within-subject experimental design. In this way, each participant acts as their own control with regard to general human performance levels.

Maturation The within-subject design, however, creates the threat of subjects becoming acquainted with the tasks and, thus, performing better with the second treatment. To mitigate this threat, we randomized the order of the DSL and the GPL treatment for each participant, which is considered to be an effective method to counterbalance the potential learning bias (Jones and Kenward 2014).

Slightly *more* than half of our sample (52.8 %) received the DSL treatment first. This does not pose a threat to our findings as it results in a bias towards the GPL treatment and our hypotheses are directed in favor of the DSL treatment.

Another threat related to the maturation of subjects is that they could become bored with the tasks, which can result in worse performance in tasks that appear later in the survey. With regard to the program comprehension tasks this threat is again mitigated by the randomization of treatment order. In addition to that, a decrease in performance could not be observed, as only one subject did not correctly complete the DSL maintenance test, which is the last test of the survey.

Mortality If the subjects who do not fully complete the survey are not representative of the total sample, the findings could be distorted. For example, if there was a significant amount of participants dropping out because they found programming with the Sprat Ecosystem DSL too hard, we would overestimate the effect of the DSL treatment. The dropout rate of 32 % we experienced is roughly equal to the average dropout rate of 30 % for online surveys (Galesic 2006). A closer examination of the dropouts reveals that most of them did not edit the task solution fields and did not spend much time on them. Additionally, the programming skill distribution of the drop-outs is very similar to the

one depicted in Fig. 5 on page 16. We therefore conclude that the drop-outs are mostly click-through participants who wanted to assess whether or not to take part in the survey and decided against it.

Instrumentation There are multiple threats related to the test instruments used in the survey. First, the type of tasks used for the program comprehension test could influence the result. We designed the tasks to be in accordance with our research goal **RG** and, thus, to cover the typical activities involved in parametrizing and maintaining an ecosystem simulation from the viewpoint of ecological modelers. To achieve this, the program comprehension tasks were created in collaboration with scientists who have worked with the Sprat Ecosystem DSL before (pilot users) so that their actual workflow is imitated when parametrizing and running high-performance ecosystem simulations with our DSL. In addition to imitating the scientists' workflow, the exercises are designed to cover all language constructs of the Sprat Ecosystem DSL.

Another concern is the comparability of the introductions to the tasks and the given code examples as well as the difficulty levels of the tasks for the DSL and the GPL treatment. To mitigate this threat, we kept all descriptions given to the subjects as short and similar as possible and had them reviewed by peers/testers beforehand. Therefore, the content of the tasks to be performed with the DSL and the GPL is the same and the information presented in the task descriptions and in the code examples is equal for the treatments (see task descriptions in the appendices).

Along similar lines, since the C++ API has also been designed by us, its quality poses an additional threat to validity. For the API design, we took the code generated by the Sprat Ecosystem DSL as a starting point and derived—again with the guidance of peers—an API in accordance with good design principles for embedded DSL (Kolovos et al. 2006). This approach of designing a C++ API that closely resembles the features of the Ecosystem DSL allowed us to make the tasks for the DSL and the GPL treatment almost identical. Making the tasks almost identical also ensures that the complexity of the solutions is similar (although the C++ solutions have more LOC).

Another threat to internal validity is posed by the choice of C++ as the GPL. We selected C++ because it is prevalent in the HPC branch of computational science (Prabhu et al. 2011; Basili et al. 2008; Faulk et al. 2009) and because we could use the code generated by the Ecosystem DSL as a starting point for the API design to keep them as comparable as possible (cf. Section 4.3). However, while C/C++ is one of the most named programming languages that participants had experience with (named 12 times), it would be interesting to investigate the effect of choosing a language that is even more popular among the studied population, such as R (named 25 times).

With regard to the DSL infrastructure maintenance test, the task chosen does not cover all the functionality of the embedded DSL-like API that is used for the configuration of the Sprat Ecosystem DSL. We argue, however, that the task is representative of the typical maintenance activities related to the configuration of the Ecosystem language.

Concerning the assessment of the participants' programming experience (Fig. 5), we rely on self-assessments instead of on pre-tests. Conducting pre-tests of programming experience would have made the survey take substantially longer, which would probably have resulted in an unacceptable drop-out rate. The results of our program comprehension experiment are not directly affected by this threat to validity because no participant could have had prior experience with the Sprat Ecosystem DSL and prior programming experience in other languages is a bias *against* our alternative hypotheses $\mathbf{H1}_1$-$\mathbf{H3}_1$. However, some users might have underestimated their Java skills, which might

affect the validity of our results concerning the infrastructure maintenance test related to $\mathbf{H4}_1$.

A last general threat that applies to all timed experiments conducted in an online survey environment is that we cannot be sure whether the subjects solved the tasks without greater interruption or distraction. We tried to mitigate this threat at least partially by measuring only the time that the web page with the survey was in focus. This, however, allows us only to correct for interruptions that occurred on the participant's computer—telephone calls, for instance, cannot be observed in this way.

External Validity

Selection To ensure that our sample is representative of the whole studied population (scientists working with ecological models from an ecological perspective), we selectively contacted institutions (i.e., research groups) consisting of relevant practitioners. There is no indication that all institutions being located in Europe or North America has an impact on the representativeness of the sample. The use of volunteers as participants could have skewed our sample as volunteers are typically more motivated than the whole population (Wohlin et al. 2012). However, as all participants receive both the DSL and the GPL treatment, they act as their own control also with regard to motivation.

Setting The generalizability of our results could be impaired by simplified tasks and a relatively short usage of the programming languages. But even though the tasks are simplified (relatively few parameters to specify etc.), they are realistic for the studied domain and they cover all functionality of the Sprat Ecosystem DSL. Nonetheless, our experiments need to be complemented by further studies that focus on the use of scientific DSL in the field. This is particularly relevant to assess the impact of the associated language tools, such as the IDE, which is replaced with a web browser-enabled editor component with less functionality (esp. lacking word completion) for this online survey with embedded experiments.

7 Conclusions

This paper presents the results of an online survey with embedded controlled experiments that analyze whether scientists from the domain of ecological modeling perform better at solving certain program comprehension tasks when using the Sprat Ecosystem DSL compared with C++. Our results show that the domain experts achieve significantly higher accuracy (from 61 % to 63 % higher average correctness point score depending on the task) and spend less time (on average 31 % up to 56 % less depedning on the task) when using our DSL instead of the comparable GPL-based solution. Additionally, the higher user ratings of the DSL compared with those of the GPL demonstrate that the scientists subjectively find it easier to work with the DSL. We believe that this is mainly due to the declarative nature of the Sprat Ecosystem DSL, which enables it to avoid some of the accidental (syntactical) complexities associated with an imperative, object-oriented language like C++. Furthermore, over 90 % of the users with only moderate Java skills are able to carry out basic maintenance tasks of the Java-based DSL infrastructure by themselves. We assume that the latter is made possible by employing an embedded DSL for configuring the Sprat Ecosystem DSL and we conclude that this is a promising method for empowering scientists to carry out basic maintenance tasks related to DSL developed for them.

Our study extends the relatively scarce body of existing evaluation research on DSL to a scientific domain with users often not specifically trained in programming and software engineering techniques. While the participants of many previous experiments are undergraduate students (e.g., Kosar et al. (2012) and Meliá et al. (2015); see Section 2), our subjects are PhD students or experts from the domain. However, since our study—like the one by Kosar et al. and that by Meliá et al.—employs simplified tasks and does not utilize full DSL tool support, our study will in the future have to be complemented by field experiments with experts as well as full tool support using tasks based on the actual work of the domain practitioners.

All in all, our results indicate that the development of further DSL for the specific needs of the ecological modeling community (and potentially also other scientific communities) should be a worthwhile investment in order to increase the productivity of scientific software developers and to enhance the reliability of their scientific results that depend on (correct) computation.

Appendix

Appendix A: Meta-Model of the Sprat Ecosystem DSL

The meta-model is shown in Fig. 9.

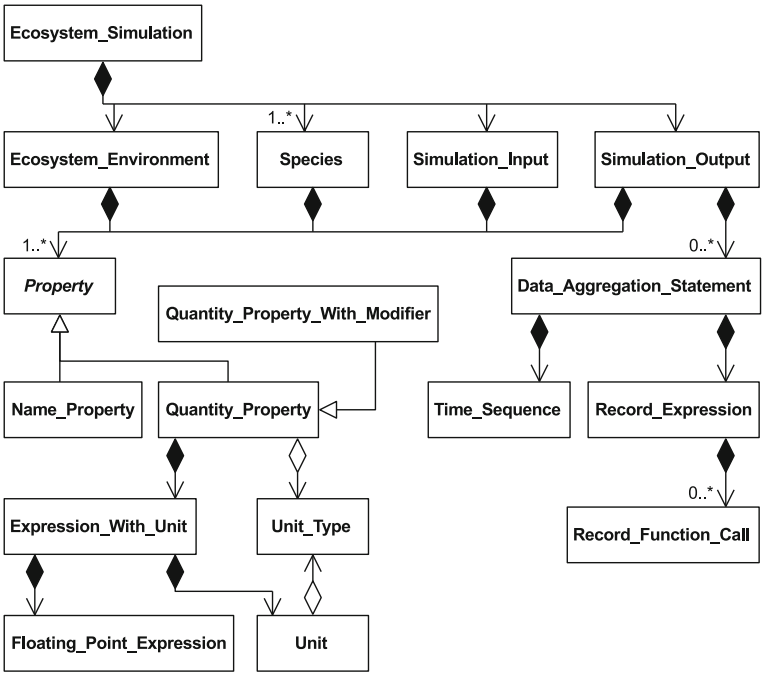


Fig. 9 UML diagram of the Sprat Ecosystem DSL meta-model

Appendix B: Academic Background Questionnaire

1. Your name [free-form question; optional]
2. Your e-mail address [free-form question; optional]
3. What is your highest educational qualification? (e.g., Bachelor, Master, Ph.D. etc.) [free-form question]
4. In which discipline did you acquire this qualification? [free-form question]
5. What is your current occupation? (e.g., bachelor/master/Ph.D. student, post-doc, professor etc.) [free-form question]

Appendix C: Programming Experience Questionnaire

1. Have you ever written a computer program using a programming language? (e.g., C, Matlab, Java, R etc.) [yes/no]
2. Have you ever been required to write a computer program for work/for your studies? [yes/no]
3. If applicable, which programming languages (including languages for special purposes like R or Matlab) have you used so far? [free-form question; optional]
4. How would you rate the level of your programming skills? [non-programmer (never written a program, difficulty with reading), beginner (problems writing easy programs, can read), regular (can write basic programs, no problems reading), advanced (can write complex programs), expert (years of experience)]
5. How experienced are you with the programming language C/C++? [non-programmer (never written a program, difficulty with reading), beginner (problems writing easy programs, can read), regular (can write basic programs, no problems reading), advanced (can write complex programs), expert (years of experience)]
6. How interested are you in programming in general? [6-point Likert scale]
7. If applicable, what was the primary source of information for learning how to program? (e.g., internet, book, a colleague/friend etc.) [free-form question; optional]
8. If applicable: I am confident that the programs I have written are correct (i.e., that they actually compute what I intended them to). [6-point Likert scale]

Appendix D: Parametrization and Recording Exercise

The exercises are shown in Figs. 10 and 11 (parametrization exercise) as well as Figs. 12 and 13 (recording exercise). Note that in the screenshots of the survey, the Sprat Ecosystem DSL is always referred to as *EcoDSL*; this was the working title of the DSL when conducting the survey.

1. Concepts of marine ecosystem simulations—such as fish species—can easily be specified with the Sprat Ecosystem DSL. [6-point Likert scale]
2. Concepts of marine ecosystem simulations—such as fish species—can easily be specified with C++. [6-point Likert scale]
3. The Sprat Ecosystem DSL seems simple to use. [6-point Likert scale]
4. C++ seems simple to use. [6-point Likert scale]
5. The Sprat Ecosystem DSL programs were easy to understand. [6-point Likert scale]

EcoDSL Parameterization Example

The following piece of code gives you an example for using EcoDSL. It specifies ecosystem parameters and introduces two fish species (cod and herring) as well as their parameters.

```
Ecosystem {
    OxygenLevel: 0.1 [mmol/l]
    WindSpeed: 12 [km/h]
}
Species cod {
    MaxMass: 1 [kg]
    IngestionExponent: 0.5
}
Species herring {
    MaxMass: 100 [g]
    IngestionExponent: 0.75
}
```

Note that the decimal separator of a rational number has to be a period ('.') and *cannot* be replaced by, e.g., a comma (',') as it is done in some locales.

Your Task

Your task is to modify the EcoDSL example in order to parameterize a simulation that features two fish species (cod and herring) and implements the ecosystem as well as the species parameters given in the following two tables.

Ecosystem parameter	**Value**
Area	1000 m^2
PredatorPreyRatio	3.5
SeabirdPredationFactor	0.025 d^-1

Species parameter	**Value for cod**	**Value for herring**
MaxLength	1.2 m	45 cm
MaxSwimmingSpeed	2.52 km/h	40 cm/s
GrowthRate	0.1 d^-1	0.15 d^-1

Fig. 10 Parametrization exercise with the Sprat Ecosystem DSL

6. The C++ programs were easy to understand. [6-point Likert scale]
7. The Sprat Ecosystem DSL seems too technical to specify concepts of marine ecosystem simulations. [6-point Likert scale]
8. C++ seems too technical to specify concepts of marine ecosystem simulations. [6-point Likert scale]
9. It is easy to introduce changes to existing Sprat Ecosystem DSL programs. [6-point Likert scale]
10. It is easy to introduce changes to existing C++ programs. [6-point Likert scale]
11. I was able to understand the meaning of the Sprat Ecosystem DSL source code quickly. [6-point Likert scale]
12. I was able to understand the meaning of C++ source code quickly. [6-point Likert scale]

C++ Parameterization Example

The following piece of code gives you an example of how to use C++ to specify ecosystem parameters (`OxygenLevel` and `WindSpeed`) as well as parameters for two fish species (`MaxMass` and `IngestionExponent`).

```
struct ModelParameters {
    static constexpr double OxygenLevel = 0.1;
    static constexpr double WindSpeed = 12.0;
    const real MaxMass[2];
    const real IngestionExponent[2];
    ModelParameters() :
        MaxMass {1.0, 0.1},
        IngestionExponent {0.5, 0.75}
    {}
};
```

Note that the decimal separator of a rational number has to be a period ('.') and *cannot* be replaced by, e.g., a comma (',') as it is done in some locales.

Your Task

Your task is to modify the C++ example in order to parameterize a simulation that features two fish species (cod and herring) and implements the ecosystem as well as the species parameters given in the two tables below.

Note: as C++ has no built-in units for numerical quantities, you have to pay close attention not to make conversion errors. You have to convert each quantity to the unit that the C++ model expects (specified in the tables below). To convert between units you can write standard arithmetic expressions such as `0.1 * 1000.0` to convert, e.g., 0.1 kilogram to gram.

Ecosystem parameter	Unit assumed in the C++ code	Value
Area	m^2	1000 m^2
PredatorPreyRatio	dimensionless	3.5
SeabirdPredationFactor	d^-1	0.025 d^-1

Species parameter	Unit assumed in the C++ code	Value for cod (species 1)	Value for herring (species 2)
MaxLength	m	1.2 m	45 cm
MaxSwimmingSpeed	m/s	2.52 km/h	40 cm/s
GrowthRate	d^-1	0.1 d^-1	0.15 d^-1

Fig. 11 Parametrization exercise with C++

13. Which difficulties did you have while using the Sprat Ecosystem DSL? [free-form question; optional]
14. Which suggestions do you have for improving the Sprat Ecosystem DSL programming language? [free-form question; optional]

Appendix E: Exercise in Modifying the Sprat Ecosystem DSL

The exercise is shown in Fig. 14.

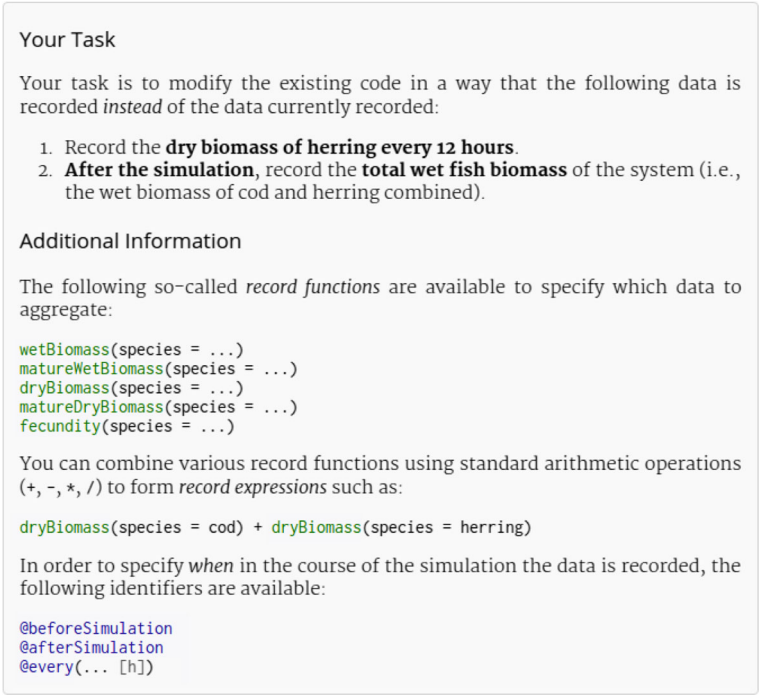


Fig. 12 Recording exercise with the Sprat Ecosystem DSL

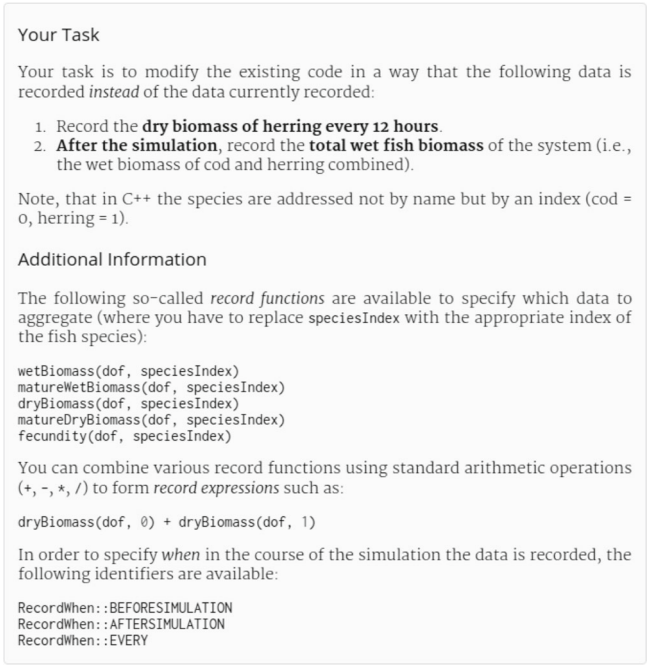


Fig. 13 Recording exercise with C++

Fig. 14 Exercise in modifying the Sprat Ecosystem DSL

1. How would you rate your expertise regarding the Java programming language? [non-programmer (never written a program, difficulty with reading), beginner (problems writing easy programs, can read), regular (can write basic programs, no problems reading), advanced (can write complex programs), expert (years of experience)]
2. The meaning of the Java code was easy to comprehend. [6-point Likert scale]
3. It was easy to recognize the statements in the Java code that were relevant for completing the task. [6-point Likert scale]

References

Almorsy M, Grundy J, Sadus R, van Straten W, Barnes DG, Kaluza O (2013) A suite of domain-specific visual languages for scientific software application modelling. In: Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2013. IEEE, pp 91–94

April A, Abran A (2012) Software Maintenance Management: Evaluation and Continuous Improvement. Wiley

Basili VR, Caldiera G, Rombach HD (1994) Goal question metric paradigm. In: Encyclopedia of Software Engineering. Wiley, pp 528–532

Basili VR, Cruzes D, Carver JC, Hochstein LM, Hollingsworth JK, Zelkowitz MV, Shull F (2008) Understanding the high-performance-computing community: a software engineer's perspective. IEEE Softw 25(4):29–36

Bortz J, Döring N (2006) Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler, 4th edn. Springer

Bortz J, Schuster C (2010) Statistik für Human- und Sozialwissenschaftler, 7th edn. Springer

Consel C, Marlet R (1998) Architecture software using a methodology for language development. In: Principles of Declarative Programming, LNCS, vol 1490. Springer, pp 170–194

Faulk S, Loh E, Vanter MLVD, Squires S, Votta LG (2009) Scientific computing's productivity gridlock: How software engineering can help. Comput Sci Eng 11:30–39

Fowler M (2010) Domain-Specific Languages. Addison-Wesley

Galesic M (2006) Dropouts on the web: Effects of interest and burden experienced during an online survey. J Off Stat 22(2):313–328

Hevner AR, Linger RC, Webb Collins R, Pleszkoch MG, Walton GH (2005) The impact of function extraction technology on next-generation software engineering. Tech. rep. Carnegie Mellon University

Hoisl B, Sobernig S, Strembeck M (2014) Comparing three notations for defining scenario-based model tests: a controlled experiment. In: 9th International Conference on the Quality of Information and Communications Technology (QUATIC), pp 95–104

Jedlitschka A, Ciolkowski M, Pfahl D (2008) Reporting experiments in software engineering. In: Shull f, Singer J, DI Sjøberg (eds) Guide to advanced empirical software engineering. Springer, pp 201-228

Johanson AN (2016) Data and scripts for the Sprat Ecosystem DSL survey. doi:10.5281/zenodo.61373

Johanson AN, Hasselbring W (2014a) Hierarchical combination of internal and external domain-specific languages for scientific computing. In: Proceedings of the 2014 European Conference on Software Architecture Workshops, ACM, ECSAW'14, pp 17:1–17:8

Johanson AN, Hasselbring W (2014b) Sprat: Hierarchies of domain-specific languages for marine ecosystem simulation engineering. In: Proceedings TMS SpringSim'14, SCS, pp 187–192

Johanson AN, Hasselbring W, Oschlies A, Worm B (2016) Evaluating hierarchical domain-specific languages for computational science: Applying the Sprat approach to a marine ecosystem model. In: Carver j, Hong NPC, Thiruvathukal GK (eds) Software Engineering for Science. Chapman and Hall

Jones B, Kenward M (2014) Design and Analysis of Cross-Over Trials. Taylor & Francis

Kieburtz R, Hook J (1995) Software design for reliability and reuse (sdrr) project phase I final scientific and technical report. Tech. rep. Pacific Software Research Center

Kieburtz RB, McKinney L, Bell JM, Hook J, Kotov A, Lewis J, Oliva DP, Sheard T, Smith I, Walton L (1996) A software engineering experiment in software component generation. In: Proceedings of the 18th international conference on software engineering (ICSE'96), pp 542–552

Knight J (2002) Safety critical systems: challenges and directions. In: Proceedings ICSE'02. IEEE, pp 547–550

Kolovos DS, Paige RF, Kelly T, Polack FA (2006) Requirements for domain-specific languages. In: Proceedings of ECOOP Workshop on Domain-Specific Program Development (DSPD)

Korman AK (1971) Industrial and Organizational Psychology. Prentice-Hall

Kosar T, Martı PE, Barrientos PA, Mernik M et al. (2008) A preliminary study on various implementation approaches of domain-specific language. Inf Softw Technol 50(5):390–405

Kosar T, Mernik M, Carver JC (2012) Program comprehension of domain-specific and general-purpose languages: Comparison using a family of experiments. Empir Softw Eng 17(3):276–304

Kosar T, Bohra S, Mernik M (2016) Domain-specific languages: a systematic mapping study. Inf Softw Technol 71:77–91

Likert R (1932) A technique for the measurement of attitudes. Arch Psychol 22(140):5–55

Meliá S, Cachero C, Hermida JM, Aparicio E (2015) Comparison of a textual versus a graphical notation for the maintainability of mde domain models: an empirical pilot study. Softw Qual J:1–27

Mernik M, Heering J, Sloane AM (2005) When and how to develop domain-specific languages. ACM Comput Surv (CSUR) 37(4):316–344

Palyart M, Lugato D, Ober I, Bruel J (2012) MDE4HPC: An approach for using model-driven engineering in high-performance computing. In: Proceedings SDL'11: Integrating System and Software Modeling, LNCS, vol 7083, pp 247–261

Prabhu P, Jablin TB, Raman A, Zhang Y, Huang J, Kim H, Johnson NP, Liu F, Ghosh S, Beardl S, Oh T, Zoufaly M, Walker D, August DI (2011) A survey of the practice of computational science. In: State of the Practice Reports, ACM, SC'11, pp 19:1–19:12

Ricca F, Scanniello G, Torchiano M, Reggio G, Astesiano E (2010) On the effectiveness of screen mockups in requirements engineering: results from an internal replication

Shapiro SS, Wilk MB, Chen HJ (1968) A comparative study of various tests for normality. J Am Stat Assoc 63(324):1343–1372

Siegmund J, Siegmund N, Apel S (2015) Views on internal and external validity in empirical software engineering. In: IEEE/ACM 37Th IEEE international conference on software engineering (ICSE 2015), pp 9-19

da Silva AR (2015) Model-driven engineering: a survey supported by the unified conceptual model. Comput Lang Syst Struct 43:139–155

van Solingen R (1999) Berghout, E. McGraw-Hill, The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development

Stahl T, Völter M (2006) Model-Driven Software development: Technology, Engineering, Management. Wiley

Webb Collins R, Hevner AR, Walton GH, Linger RC (2008) The impacts of function extraction technology on program comprehension: a controlled experiment. Inf Softw Technol 50(11):1165–1179

Wilcoxon F (1945) Individual comparisons by ranking methods. Biom Bull 1(6):80–83

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer

Arne N. Johanson received a Ph.D. in computer science from Kiel University, Germany in 2016. He was awarded a scholarship from the Helmholtz Research School Ocean System Science and Technology (HOSST). His research focuses on adapting software engineering techniques for computational science. Besides studying the potential of domain-specific languages in this context, his research interests also include ecological modeling and data mining for ecological research.

Wilhelm Hasselbring is professor of Software Engineering at Kiel University, Germany. In the competence cluster Software Systems Engineering (KoSSE), he coordinates technology transfer projects with industry. In the excellence cluster Future Ocean, a large-scale collaborative project of Kiel University and the GEOMAR Helmholtz Centre for Ocean Research Kiel, he is principal investigator and coordinator of the research area Ocean Observations. He is principal investigator of the Helmholtz Research School Ocean System Science and Technology (HOSST). His research interests include software engineering and distributed systems, particularly software architecture design and evaluation. He received his Ph.D. in Computer Science from the University of Dortmund, Germany. He is a member of the ACM, the IEEE Computer Society, and the German Association for Computer Science.