Empir Software Eng (2018) 23:2734–2763
https://doi.org/10.1007/s10664-017-9593-2

Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments

Tomaž Kosar[1] · **Sašo Gaberc**[1] · **Jeffrey C. Carver**[2] · **Marjan Mernik**[1]

Published online: 9 February 2018

Abstract Domain-specific languages (DSLs) allow developers to write code at a higher level of abstraction compared with general-purpose languages (GPLs). Developers often use DSLs to reduce the complexity of GPLs. Our previous study found that developers performed program comprehension tasks more accurately and efficiently with DSLs than with corresponding APIs in GPLs. This study replicates our previous study to validate and extend the results when developers use IDEs to perform program comprehension tasks. We performed a dependent replication of a family of experiments. We made two specific changes to the original study: (1) participants used IDEs to perform the program comprehension tasks, to address a threat to validity in the original experiment and (2) each participant performed program comprehension tasks on either DSLs or GPLs, not both as in the original experiment. The results of the replication are consistent with and expanded the results of the original study. Developers are significantly more effective and efficient in tool-based program comprehension when using a DSL than when using a corresponding API in a GPL. The results indicate that, where a DSL is available, developers will perform program comprehension better using the DSL than when using the corresponding API in a GPL.

Keywords Domain-specific languages · General-purpose languages · Program comprehension · Controlled experiment · Replication

Communicated by: Sven Apel

The authors acknowledge the financial support from the Slovenian Research Agency (research core funding No. P2-0041).

✉ Tomaž Kosar
tomaz.kosar@um.si

[1] Faculty of Electrical Engineering and Computer Science, University of Maribor, Koroška cesta 46, 2000 Maribor, Slovenia

[2] Department of Computer Science, University of Alabama, Tuscaloosa, AL, USA

1 Introduction

A domain-specific language (DSL) (Mernik et al. 2005; van Deursen et al. 2000) is a programming language that allows developers to express concepts in terminology drawn from a particular domain (Sprinkle et al. 2009). DSLs, an increasingly popular software development technique, use concepts from the problem domain rather than the solution domain (Goulão et al. 2016). Conversely, the more widely known general-purpose languages (GPLs) do not target any specific domain and can be used for many different domains. When using a GPL, developers often make use of APIs developed for specific domains. For the remainder of this paper, when use the term 'GPL' as a shorthand to refer to the domain-specific API provided by the GPL. Because DSLs express domain concepts more effectively, developers often use them to address the complexity of GPLs (Wile 2001; Mauw et al. 2004). Many argue that DSLs increase the quality of the delivered code (Hermans et al. 2009).

Evidence from empirical studies can help increase the usage of DSLs by providing important information about their benefits, challenges, and shortcomings. Two recent studies, which surveyed papers from 2006 through 2012, found a very small percentage of DSL studies contain controlled experiments – 1.3% by one study (Kosar et al. 2016) and 4% by another study (Carver et al. 2011). These studies highlight the need for experiments about DSLs. Barišić et al also highlight the need for more studies to evaluate the quality and usability of DSLs (Barišić et al. 2017). In addition, there is a lack of evidence for comparing program comprehension tools developed for DSLs (Cornelissen et al. 2011).

Recently, the number of empirical studies about DSLs has increased (Häser et al. 2016; Hoisl et al. 2014; Albuquerque et al. 2015; Meliá et al. 2016; Umuhoza et al. 2015). It is important to continue expanding the body of literature with empirical studies that focus on different research goals. For instance, a recent study by Johanson and Hasselbring (2016) complements the findings from Kosar et al. (2012) by investigating programmers with only moderate experience as opposed to Kosar et al.'s study where programmers were from technical domains.

In addition, study replications can increase the trustworthiness of the results from empirical studies. Our original family of controlled experiments that compared users' program comprehension between DSLs and GPLs programs showed that developers were more effective and efficient with DSLs than with GPLs (Kosar et al. 2010; Kosar et al. 2012). To expand upon those results, we performed a replication with some specific changes to the experimental setting to strengthen the original results. The goal of the replication is *to increase the value of the original study results by replicating with tool support.* While the long-standing belief that DSLs make developers more efficient than GPLs (Consel and Marlet 1998) has been taken for granted (Prähofer et al. 2013; Varanda Pereira et al. 2016), the lack of evaluation research suggests that a more skeptical view is appropriate. Therefore, we are particularly interested in providing more empirical evidence about the relative accuracy and efficiency of developers using a DSL compared with developers using a GPL.

Next, we describe the original study in more detail. Then, we present the motivation for the replication described in this paper.

1.1 The Original Study

In 2012, three of the four authors of the current paper conducted a family of experiments comparing the participants' comprehension of DSL programs against their comprehension

of GPL programs (Kosar et al. 2012). The goal of the experiments was to provide empirical data regarding the common belief in the DSL community that DSLs users are more efficient than GPL users (Hudak 1998; van Deursen and Klint 1998; Consel and Marlet 1998). We focused on program modification tasks during the implementation and evolution phases of the software lifecycle, both of which require program comprehension (Storey 2005; Hevner et al. 2005). Three types of cognitive activities are fundamental during program modification: *learnability*, *understandability*, and *changeability* (Jung et al. 2004). These three activities are strongly influenced by the extent to which a programmer's mental model of the software matches and predicts the actual behavior of the software. We tested two hypotheses:

$H1$
: *The use of a DSL significantly increases the correctness of the participants' program comprehension over the use of a GPL.*

$H2$
: *The use of a DSL significantly improves the efficiency of the participants' program comprehension over the use of a GPL.*

To test the effects of the DSLs, we performed three identical experiments, each using a different domain: feature diagrams (FD), graph descriptions (GD), and graphical user interfaces (GUI). Because there was not a mature IDE for the chosen DSL (Chiş et al. 2015), the participants performed all tasks manually, on paper. Each experiment involved at least 34 participants.

We first conducted a training session to introduce domain concepts and to notations (DSL and GPL) to the participants. Next, each participant performed a program comprehension task. We assigned half of the participants to use a DSL and half to use a GPL (to balance the order). Then, each participant performed a second program comprehension task on the other language (DSL or GPL). For each program comprehension task, each participant reviewed a segment of code and then answered a series of questions to measure his or her level of understanding of that code. The questionnaire included a set of questions on *learning*, a set on *understanding*, and a set on *evolving*. Finally, the participants completed a feedback survey.

The results of the original study showed support for both hypotheses. The participants' correctness was significantly higher for DSL programs than for GPL programs ($H1$). The participants were significantly more efficient on DSL programs than on GPL programs ($H2$).

1.2 The Motivation for the Replication

As stated earlier, traditionally DSL researchers report on the design and implementation of a particular DSL, rather than performing empirical validations of those DSLs (Kosar et al. 2016). This situation is similar for replications of DSL studies. We found no examples of replicated studies on DSLs. Our motivation for conducting this replication is three-fold:

1. To understand whether the results of the original study held with a new set of participants;
2. To address some of the limitations of the original study; and
3. To conduct the first replicated study for program comprehension in DSLs.

One of the primary threats to validity in the original study was that the participants performed the program comprehension tasks manually (i.e. on paper, without tool support).

A more realistic scenario is when participants perform program comprehension tasks supported by tools in integrated development environments (IDEs). Therefore, in the replication, described in Section 3, the participants perform the program comprehension tasks using IDE support on a computer.

1.3 Summary

The main contributions of this paper are:

- description of a replication of a study comparing DSLs to GPLs,
- insight into the relative correctness and efficiency of developers when using DSLs, compared with GPLs, and
- integration of IDE tools into a program comprehension study.

The remainder of this paper is organized based on replication reporting guidelines (Carver 2010). Section 3 describes the replication. Section 4 presents the replication results and data analysis. Section 5 compares the results of the replication to those of the original study. Section 6 describes threats to validity. Section 7 summarizes the key findings and outlines the future work.

2 Related Work

Historically, published DSL research has focused on the design and implementation of DSLs rather than on empirical validation of the usefulness and efficacy of DSLs (Kosar et al. 2016). More recently, researchers have begun publishing empirical studies of DSLs. To properly position our own empirical research, this section summarizes those studies, some of which we mentioned in the introduction.

A 2016 systematic mapping study found that DSL designers rarely validate whether the new DSL meets the needs of its target users (Kosar et al. 2016). Because DSL designers build DSLs for a specific domain, Barišić, et al. note that it is important for DSL designers to evaluate the usability of the DSL in that domain. They design a DSL evaluation approach that includes usability as a key concept and illustrate this approach via an experiment comparing the usability of Pheasant (PHysicist's EAsy Analysis Tool) with the usability of a pre-existing C++ solution. The results showed that users were more effective with Pheasant (Barišić et al. 2017).

In an experiment inspired by our original family of experiments (Kosar et al. 2012), Johanson et al. conducted a family of experiments in the ecology domain comparing the Sprat Ecosystem DSL to its GPL counterpart. The study results, similar to those from our original study, showed that users solved program comprehension tasks more accurately (61% to 63% higher) and using less time (from 31% to 56% less depending on the task) with the DSL than with the corresponding GPL (Johanson and Hasselbring 2016). The key differences between this study and our original study help increase the external validity of our results. First, Johanson et al. performed their study in a less technical domain (ecology) with end-users who had less programming skills compared with the domain and users in our original study, suggesting that the results may transfer across domains (although more study is needed to confirm this observation). Second, Johanson et al. allowed participants to use computers during their study whereas our study was paper-based. In our replication described in the paper, we followed this example by allowing participants to perform the study tasks on computers. We extend Johnson et al.'s

study by encouraging participants to use the full tool support provided by the respective IDEs.

According to Kosar et al. DSL maintenance is an insufficiently investigated aspect of the DSL development process (Kosar et al. 2016). In one counterexample to this observation, Häser et al. extended a Behavior Driven Development DSL with business domain concepts. They then performed a study to compare participants' performance when creating test case specifications using the extended DSL with their performance when using the original DSL (without the modifications). The results showed that those who used the modified DSL were significantly faster in creating the test specification without lowering the perceived quality.

The designer's choice of graphical vs. textual notation is another important factor in DSL design. Both notations have strengths and limitations. While empirical studies can provide insight into the usability aspects of each notation, DSL designers often choose their notation based upon intuition rather than upon such empirical data. Two empirical studies highlight some of the issues related to notation that could be considered DSL designers. First, Melia et al. show that while end-users are significantly more effective and efficient with a textual DSL than with a graphical DSL, they do not have a preference between the two notations (Meliá et al. 2016). Second, Hoisl et al. compared the performance of professional software engineers when validating models presented in three formats: a semi-structured natural language, a diagrammatic notation, and a fully structured textual notation. The results showed that the semi-structured natural language resulted in the highest correctness and the least amount of time (Hoisl et al. 2014).

3 The Replication

In this section we describe the design and execution of the replication. In the replication we reused the hypothesis, lab package, and data collection instruments from the original study, with some slight changes (as detailed in the following subsections).

3.1 Overview

Based on our motivation for conducting the replication (Section 1.2), we made the following minor changes to the original study design, which we describe in the remainder of this section:

- The participants use *IDEs* to perform the program comprehension tasks, rather than performing them *manually*;
- We use a *between-subjects* rather than *within-subjects* study design;
- The data collection includes information about the types of IDE tools participants use for the program comprehension tasks.
- A new experimenter joined the team.

There is an ongoing debate within the software engineering community regarding the level of interaction between original and replicating researchers (Kitchenham 2008; Shull et al. 2008). In the spirit of full disclosure, this replication can be classified as a "dependent replication" (Baldassarre et al. 2014), where the same team of experimenters who conducted the original study also conducted the replication (with the addition of one new team member). We based the replication off of the lab package from the original study. We encourage others to replicate our study and will provide support and materials, as needed. We have

made a replication package, containing all relevant materials, available for download[1]. To preserve the integrity of future replications (because the replication package has the answers to all of the questionnaires) we have password protected this file. The password is available upon request from the authors.

3.2 Experimental Design

This section describes the study along with a justification for the specific changes made to the original study design.

3.2.1 Research Questions & Hypotheses

For the replication, we reused the research questions from the original study:

$RQ1$: **Does program comprehension increase when programmers use DSLs instead of GPLs?**

$RQ2$: **Does efficiency increase when programmers use DSL programs instead of GPL programs?**

These research questions led to the same hypotheses as in the original study:

$H1$ *The use of a DSL significantly increases the correctness of the participants' program comprehension over the use of a GPL.*

$H2$ *The use of a DSL significantly improves the efficiency of the participants' program comprehension over the use of a GPL.*

Based on results from our original study (Kosar et al. 2012), we had reason to expect that DSLs would outperform GPLs. Therefore, we stated directional hypotheses (i.e. in the direction of the DSLs). Given that the hypotheses were directional, one-tailed statistical tests are the most appropriate and were used during our statistical analysis.

3.2.2 Experimental Steps

To address the research questions and test the hypotheses, we followed the same steps as the original experiment (Kosar et al. 2012). In this section we provide an overview of the experimental steps along with a brief description of the changes made in the replication.

Table 1 lists the steps followed by the study participants. During the *Training Session*, we introduced the basic idea and concepts from the problem domains (FD, GD, or GUI, see Table 2) to the participants via an example application. Appendix A includes an example of a slide with typical DSL program used during training session. Next, we gave the participants a tutorial in their assigned language type (DSL or API in the GPL) along with an example program. Finally, we provided a very basic introduction to the IDE tools used during the experiment. We grouped the IDE tools as follows:

– *Build* – including debuggers, watch, variable lists, problems (errors and warnings), and console;

[1] https://lpm.feri.um.si/projects/ReplicationLabPackage.zip

Table 1 Replication design

Procedure	Steps						Execution time
	FDL		GD		GUI		
	DSL	GPL	DSL	GPL	DSL	GPL	
	27*	24	27	25	20	23	
Training	Problem domain presentation						30 min
	Tutorial on notation						
	IDE tools demonstration						
Background	Survey						5 min
Material distribution	Training slides						10 min
	Program comprehension questionnaire						
	Background and feedback questionnaire						
Questionnaire execution	22 program comprehension questions						120 min
	Tool usage reports						
Feedback	Survey						15 min

*Number of participants

- *Editor* – i.e., functionalities typical for source code editors, such as syntax highlighting, indentation, autocomplete, bracket matching functionality;
- *Find & Replace* – note, normally this tool could be categorized as an *editor* tool, however we suspected that this tool will be quite frequently used, therefore we decided to provide it as a separate option;
- *Manual Program Comprehension* – i.e. no IDE tool was used; and
- *Other* – a tool not included in the list.

We added this last training in the replication. The tutorials we used for training are part of the lab package described in Section 3.1.

During the *Background Survey*, the participants answered demographic questions. In the replication, we added a question regarding the participants' experience with the IDEs used in the study.

Table 2 Domain descriptions

Experiment	Domain	Description	Source
1	FD	Feature diagrams provide the ability to define variable and fixed parts of a given system configuration.	lab package
2	GD	Graph descriptions provide the ability to define graphs using two main concepts: nodes and edges.	www.graphviz.org
3	GUI	Graphical user interfaces domain provide the ability to define computer interfaces using components, e.g., forms, panels, tables, buttons, canvases.	MS Visual Studio

During *Materials Distribution*, we provided the participants with the artifacts required for the study (slides from training session, the tutorial on language type, and the program comprehension questionnaire). Because the participants performed the replication activities on the computer rather than on paper, we distributed fewer hard-copy materials during the replication (because we provided some materials electronically).

During the *Questionnaire Execution* phase, the participants answered 22 questions. To reduce the potential for domain bias, the participants used programs from the three problem domains described in Table 2, Feature Diagrams (FD), Graph Descriptions (GD), and Graphical User Interface (GUI), each in separate experiment. For each question on the program comprehension questionnaire the participants reported which group(s) of IDE tools they used (i.e. Build, Editor, Find & Replace, Manual, Other) To reduce the chances of inconsistent tool usage reporting, we instructed the participants on which tools belonged to each tool type (i.e. build, editor, find & replace, manual, or other) and provided a multiple choice question to facilitate consistent reporting.

Finally, during the *Feedback Survey*, the participants provided feedback regarding: the complexity of the exercises and the simplicity of using the DSL/GPL libraries. The first question was to identify whether we introduced a threat to validity with different DSL/GPL questionnaires. The second question was to understand the participants' opinion on programming with DSL/GPL library.

We made two key changes to the design of the original study. First, rather than reviewing code on paper, the participants reviewed code in an IDE. We instructed the participants on which IDE to use for each questionnaire. Note that while the participants could use various IDE tools to help them understand the code, they were not allowed to check the output of code execution or use other tools (e.g. other IDEs or the Internet). Second, each participant reviewed either the GPL code or the DSL code (not both).

3.2.3 Question Design

Table 3 provides a detailed description of the questionnaire design (all questionnaires are available online[2]). We used the same design for the GPLs and the DSLs. The first column of the table indicates the cognitive activity studied by the question, defined as follows:

Learn: Questions about the programmer's ability to learn how to use a language or API.
Understand: Questions about the programmer's ability to understand an existing program.
Evolve: Questions about the programmers ability to modify a program given changed requirements.

We chose these three activities, because they cover the majority of program comprehension activities. An industrial case study showed that programmer spend more than 28% of their time in learning and understanding (Hevner et al. 2005).

The column "Question Focus" further decomposes the cognitive activities. Questions 1-16 were multiple choice questions that had one correct answer. Questions 17-22 required the participants to modify code. The course instructors examined the submitted code to determine its correctness. Appendix B includes an example of a typical DSL program from each domains along with their respective outputs.

[2] http://lpm.feri.um.si/projects/ReplicationQuestionnaires.zip

Table 3 Description of questions

Cognitive activity	Question	Focus	Question content	Answer type
Learn	1 & 2	Syntax	Select the statement or program with the correct syntax	Multiple choice - choose 1
	3 & 4	Semantics	Select the statements or programs that do not make sense	
	5 & 6	Meaning	Select the program that produces the given configuration (FD) or figure (GD & GUI)	
Understand	7 & 8	Overall result	Select valid configuration (FD) or figure (GD & GUI) produced by a given program	
	9 & 10	Specific concepts	Identify specific language constructs described by a given program	
	11 & 12	Compare	Select the program that produces the given result	
	13 & 14	Consistency	Select a valid configuration (FD) or figure (GD & GUI) produced by a new language construct	
	15 & 16	Comment	Read comments in a more complicated program and select correct result	
Evolve	17 & 18	Expand	Expand the given program by adding new functionality	Revised code
	19 & 20	Remove	Remove functionality from the given program	
	21 & 22	Change	Change the functionality of a given program	

3.2.4 Variables

Based on the research questions and experimental steps, we defined the independent and dependent variables (Table 4). All of the variables, except for *Experience with IDE* are common between the original study and the replication. Because the replication participants performed the program comprehension tasks using an IDE, we added *Experience with IDE* to determine whether there was any effect due to this experience.

While most of the variables are self-explanatory, here we explain those that are not. *Question type* defines whether the question asked about learnability, understandability, or changeability. Each questionnaire covered all three cognitive activities.

Efficiency accounts for both correctness and time because using only time or only performance would provide questionable results (Nugroho 2009).

Simplicity of Use refers to the participants' satisfaction with the DSL or the GPL library. This variable checks whether participants' opinion supports the findings from the correctness, time, and efficiency variables. The participants provided this information on the Feedback Survey.

Table 4 Independent and dependent variables

	Variable	Value	Reused
Independent	Language type	DSL or GPL	yes
	Domain	FD, GD, or GUI	yes
	Question type	Learn, Understand, Evolve	yes
	Domain Experience	$1-5$ Scale*	yes
	IDE Experience	$1-5$ Scale*	no
Dependent	Correctness	% correct answers	yes
	Time	Questionnaire completion time	yes
	Efficiency	Correctness/time	yes
	Simplicity of use	$1-5$ scale^{+}	yes
	Questionnaire complexity	$1-5$ scale^{-}	yes

*1 = no experience ... 5 = expert

+1 = very hard... 5 = very easy

−1 = not complex... 5 = complex

Questionnaire complexity refers to the participants' perception of how complex the various questionnaires were. The participants provided this value during the Feedback Survey. Within the same domain, we defined different tasks for the DSL and GPL questionnaires, resulting in a potential construct validity threat. Even with the effort devoted to developing equivalent questionnaires (Kosar et al. 2012), we cannot be sure of their comparability.

3.3 Participants

Similar to the original study, the replication participants were students from the Faculty of Electrical Engineering and Computer Science in Maribor, Slovenia. The participants were enrolled in undergraduate Computer Science and Information Technologies courses on Programming Languages (third year) and Generative Methods (fourth year). Because the Slovenian government encourages students to work while in school, many of the participants in our study had some level of industrial experience. In both studies students were familiar with DSLs, compilers, and some of the domains used in the experiments. The students in both studies followed the same steps.

Similar to the original study, the participants voluntarily chose whether to participate. Those that chose to participate received up to a 10% bonus on their course grade. The exact amount of the bonus depended upon the correctness of their answers. We chose to base the bonus on correctness to help motivate the participants to devote adequate effort to the tasks.

As shown in Table 1, the replication included 146 data points from 61 unique participants divided across 3 domains (FD, GD, and GUI) and 2 language types (DSL and GPL). Half of the participants were in their third year of study and half in their fourth year. To reduce any selection bias, we randomly assigned each participant to either the DSL or the GPL condition. Because some participants only attended one or two of the domains, the number of participants is not consistent across domains. This number of participants is larger than in the original study, which included 108 data points in all three experiments. Note, that in the original experiments, the participants reviewed both DSL and GPL programs.

3.4 APIs and IDEs

In Table 5 we show the GPL APIs and IDEs used for each programming task. Note that for the FD and GUI domains, the participants used the same IDE for both the GPL and the DSL languages, while in the GD domain the IDE differed between the GPL and the DSL languages. We chose these IDEs based upon their support of the programming language and the specific API. We also consider other factors when choosing the IDEs. For example, because we were unable to prevent participants from running the XAML and C# Forms programs in Visual Studio, we chose the SharpDevelop IDE instead. Additionally, we had to develop tool support (e.g. syntax highlighting, autocomplete, debugging, watch list, and error list) in Eclipse IDE for the FD DSL since it was not available by default. In all cases, we configured the IDEs to prevent participants from seeing the output of the execution.

3.5 Experiment Execution

We conducted the experiments at University of Maribor between January 2015 and April 2015. We conducted both the DSL and GPL questionnaires in the same week during the practical part of the courses. The lab course lasted three hours, which was adequate time for the participants to complete the study. Finally, we presented each participant with his or her own score in the experimental tasks (but not the overall study results).

4 Replication Results

In Section 4.1, we describe the analysis of participant experience. In Section 4.2, we describe the main results relative to the study hypotheses. In Section 4.3, we provide the results by individual question and question type. In Section 4.4, we describe which tools participants used for each task (the extension of the original study implemented in this replication). In Section 4.5, we describe the participant feedback. For all statistical analyses, we used $\alpha = 0.05$ to judge significance.

4.1 Participant Experience

To guard against any bias resulting from differing experience levels between the DSL group and the GPL group, we gathered the participants' *domain experience* and *IDE experience* (new in the replication). In Table 6 we show a comparison of domain experience between the participants from each domain and language type combination. Because the replication

Table 5 APIs and IDEs used for each programming task

Domain	Language	API	IDE
FD	GPL	Java FDL API (In Replication Package)	Eclipse
	DSL	FDL (van Deursen and Klint 2002)	Eclipse
GD	GPL	C GD API[a]	Visual studio
	DSL	DOT language (Gansner et al. 2009)	Eclipse
GUI	GPL	C# Forms (Williams 2002)	SharpDevelop
	DSL	XAML (MacVittie 2006)	SharpDevelop

[a] http://www.graphviz.org/pdf/libguide.pdf

Table 6 Participant domain experience

		N	Mean	Std. Dev.	Median	Mean rank	Z	p-value*
FD	DSL	27	2.07	1.04	2	25.89	−0.590	0.953
	GPL	24	2.08	1.02	2	26.13		
GD	DSL	27	2.37	1.25	2	26.17	−0.171	0.864
	GPL	25	2.44	1.29	3	26.86		
GUI	DSL	20	3.9	1.02	4	20.65	−0.712	0.476
	GPL	23	4.13	0.76	4	23.17		

*Mann-Whitney U test

used a "between-subjects" design and the data were not normally distributed (based upon the Shapiro Wilk normality test), we used the Mann-Whitney U test to compare domain experience. The results show no significant difference between domain experience with GPL and DSL for any of the three domains.

In Table 7 we show the participants' experience with the IDEs for each of the three domains. The only significant difference was that participants in the GD domain were more experienced with the GPL IDE than with the DSL IDE. Because our hypothesis is in favor of the DSL language type, this difference does not pose a threat. If anything, it strengthens any conclusion we might draw.

4.2 Analysis of Hypotheses

This section describes the results relative to the two study hypotheses.

4.2.1 H1: Correctness

In Table 8 we show the statistical comparison of correctness between the participants who used the GPL and those who used the DSL for each domain, using the Mann-Whitney U test. In this case, the hypothesis was one-tailed (i.e. in favor of the DSL). The results show that for all three domains, the participants using the DSL had significantly higher correctness than those using the GPL. These results provide support for H1.

4.2.2 H2: Efficiency

In Table 9 we show the statistical comparison of the efficiency between the participants who used the GPL and those who used the DSL for each domain, again using the

Table 7 Participant experience with IDEs

		N	Mean	Std. Dev.	Median	Mean rank	Z	p-value*
FD	DSL	24	2.92	0.83	3	23.48	−0.012	0.990
	GPL	22	2.86	1.04	3	23.52		
GD	DSL	22	3.18	0.91	3	18.05	−2.666	**0.008**
	GPL	23	3.78	0.85	4	27.74		
GUI	DSL	17	2.71	1.40	3	19.88	0.059	0.953
	GPL	22	2.73	1.24	3	20.09		

*Mann-Whitney U test

Significant results are highlighted in bold

Table 8 Correctness results

		N	Mean	Std. Dev.	Median	Mean rank	Z	p-value*
FD	DSL	27	83.67	10.01	84.09	31.94	−3.036	**0.002**
	GPL	24	70.26	16.25	67.05	19.31		
GD	DSL	27	85.18	10.29	86.36	33.74	−3.602	**0.000**
	GPL	25	72.00	12.90	72.73	18.68		
GUI	DSL	20	82.04	10.58	84.09	26.45	−2.211	**0.027**
	GPL	23	75.10	8.09	72.73	18.13		

*Mann-Whitney U test
Significant results are highlighted in bold

Mann-Whitney U test. The results show that for all three domains, the participants using the DSL were significantly more efficient than those using the GPL. These results provide support for H2.

4.2.3 Summary of Hypotheses Results

With these results on both hypotheses H1 and H2, we found that developers are significantly more accurate and efficient in tool-based program comprehension when using a DSL than when using a GPL.

4.3 Individual Question Analysis

Following the original study, we divided the questions into three sets based upon the three program comprehension tasks: learning, understanding, evolution. To evaluate whether language type affected the participants' performance on each type of task, we compared them separately. In Table 10 we show these results.

Examining the "Mean" values shows that participants using the DSL outperformed those using the GPL on all domains and program comprehension tasks except for the *understand* questions in the GUI domain. The Mann-Whitney U test results showed that only five of the nine differences are statistically significant. The only consistent result is that the participants performed significantly better on the *evolve* program comprehension questions across all three domain.

An examination of each question in detail provides more insight into this result. In Table 11 we present the average correctness scores for each question across all

Table 9 Efficiency Results

		N	Mean	Std. Dev.	Median	Mean rank	Z	p-value*
FD	DSL	27	1.46	0.34	1.35	37.30	−5.756	**0.000**
	GPL	24	0.81	0.26	0.83	13.29		
GD	DSL	27	1.65	0.37	1.62	38.20	−5.789	**0.000**
	GPL	25	0.92	0.23	0.86	13.86		
GUI	DSL	20	1.19	0.21	1.16	29.38	−3.593	**0.000**
	GPL	23	0.98	0.17	0.97	15.59		

*Mann-Whitney U test
Significant results are highlighted in bold

Table 10 Correctness of learn, understand, and evolve questions

			N	Mean	Std. Dev.	Median	Mean rank	Z	p-value*
FD	L	DSL	27	90.12	11.56	100.0	33.65	−4.166	**0.000**
		GPL	24	74.30	10.96	83.33	17.40		
	U	DSL	27	75.37	17.65	80.00	29.37	−1.724	0.085
		GPL	24	62.92	25.96	65.00	22.21		
	E	DSL	27	91.05	10.06	91.67	30.83	−2.557	**0.011**
		GPL	24	78.47	19.34	83.33	20.56		
GD	L	DSL	27	75.92	19.79	83.33	28.91	−1.222	0.222
		GPL	25	68.00	24.49	66.67	23.90		
	U	DSL	27	90.00	11.09	90.00	34.22	−3.925	**0.000**
		GPL	25	74.00	15.28	70.00	18.16		
	E	DSL	27	86.42	12.26	83.33	32.98	−3.423	**0.001**
		GPL	25	72.67	15.12	66.67	19.50		
GUI	L	DSL	20	81.66	14.20	83.33	24.03	−1.042	0.298
		GPL	23	75.36	18.71	83.33	20.24		
	U	DSL	20	76.50	15.99	80.00	22.03	−0.130	0.990
		GPL	23	77.39	10.54	80.00	21.98		
	E	DSL	20	91.67	11.47	100.00	29.75	−3.936	**0.000**
		GPL	23	71.01	16.06	66.67	15.26		

L - Learn, U - Understand, E - Evolve

*Mann-Whitney U test

Significant results are highlighted in bold

domain/language combinations. According to these results, the participants using the DSL were correct more often in 26 out of 33 instances, suggesting that the DSL's were easier to understand.

Table 11 Average correctness by question

	FD		GD		GUI	
	DSL	GPL	DSL	GPL	DSL	GPL
Q1	100%	100%	100%	90%	100%	86%
Q2	92%	64%	51%	40%	55%	63%
Q3	77%	58%	75%	67%	90%	76%
Q4	93%	69%	100%	90%	92%	93%
Q5	57%	52%	87%	76%	72%	56%
Q6	87%	58%	85%	57%	60%	56%
Q7	90%	65%	98%	63%	85%	91%
Q8	48%	68%	79%	73%	72%	89%
Q9	87%	79%	98%	73%	90%	84%
Q10	88%	91%	90%	69%	87%	63%
Q11	96%	64%	70%	73%	97%	65%
Average	83%	70%	85%	70%	82%	75%

4.4 IDE Tool Usage

By adding the use of IDEs to the replication, we can also examine which specific IDE tools the participants use when working with DSLs and with GPLs. We show the overall results for tool usage in Table 12. Column N_1 shows the number of participants who used each language type in each domain. Column N_2 shows the total number of responses given by those participants regarding the tools they used to answer all of the program comprehension questions. Note that for each question, participants could select more than one IDE tool or they could select zero tools.

The results in column N_2 show that participants used IDE tools more frequently when performing tasks using GPLs than when using DSLs. One possible explanation for this result is that because DSL programs are generally shorter, participants had less need for additional tools to support program comprehension. Even so, the number of times that participants used tools when working with DSLs could mean that IDE tools are important for supporting program comprehension in DSLs. Unfortunately, DSL builders often neglect to build tool support, since it takes precious development time.

In addition to the total number of times participants used the tools, it is also important to understand whether participants use different tools depending upon the type of language. The remainder of the columns in Table 12 show the distribution of the specific types of tools (*Build*, *Find & Replace*, *Editor*, and *Other*) the participants used for each language type and each domain. Comparing the distributions for each domain using the χ^2 test shows that in all three cases, participants use different tools when working with a DSL than when working with a GPL. It is interesting to note that while there are differences in each domain, there does not seem to be a pattern across domains.

To gain more insight into these results, we analyzed the impact of tools on each question type (*Learn*, *Understand*, and *Evolve*). We can make some interesting observations from the distribution of tool usage for each question type shown in Table 13. First, for all three domains, the participants used the *build* tools most frequently for the *learn* question type. This result make sense because the *build* functionality helps a developer identify syntax errors. When a developer is learning a new domain, he/she often uses build and execute functionalities to check the result of the program. After learning the domain, the developer uses build tools less often and relies more on her/his mental model of the program. Conversely, for the *understand* and *evolve* question types, the participants used *build* tools very infrequently. Second, for all three domains, the participants used the *editor* and *Find & Replace* tools most frequently for the *evolve* question type. This result makes sense because these questions require the participants to modify the program code itself.

Unlike the statistical results for the overall distribution of tool usage (Table 12), we do not observe a consistent pattern for the χ^2 results for individual question types.

Table 12 Distribution of IDE tool usage

		N_1	N_2	Build	F & R	Editor	Other	p-value*
FD	DSL	27	157	22%	8%	33%	35%	**<.001**
	GPL	24	256	19%	39%	27%	14%	
GD	DSL	27	218	19%	34%	28%	17%	**.010**
	GPL	25	268	23%	22%	38%	15%	
GUI	DSL	20	168	23%	37%	39%	0%	**<.001**
	GPL	23	231	20%	46%	24%	7%	

N_1 – number of participants, N_2 – number of received answers

F & R – find and replace

*χ^2 test

Significant results are highlighted in bold

Table 13 Distribution of IDE tool usage of learn, understand, and evolve questions

			N	Build	F & R	Editor	Other	p-value*
FD	L	DSL	66	44%	0%	41%	15%	**0.000**
		GPL	62	39%	23%	23%	16%	
	U	DSL	36	0%	14%	14%	72%	**0.000**
		GPL	84	11%	43%	23%	24%	
	E	DSL	55	13%	15%	38%	15%	**0.000**
		GPL	110	15%	45%	34%	6%	
GD	L	DSL	61	56%	5%	23%	16%	0.622
		GPL	79	54%	8%	28%	10%	
	U	DSL	55	0%	29%	36%	35%	0.456
		GPL	84	0%	38%	36%	26%	
	E	DSL	102	9%	55%	27%	9%	**0.000**
		GPL	105	20%	20%	49%	11%	
GUI	L	DSL	51	73%	4%	24%	0%	**0.023**
		GPL	60	55%	23%	20%	2%	
	U	DSL	45	0%	60%	40%	0%	0.079
		GPL	73	0%	75%	25%	0%	
	*E	DSL	72	3%	47%	50%	0%	0.074
		GPL	74	11%	53%	36%	0%	

L - Learn, U - Understand, E - Evolve

*χ^2 test

Significant results are highlighted in bold

For the FD domain, there is a significantly different distribution of tools for DSL and GPL for all three question types. Conversely, for the other two domains, the distributions are significantly different for only one question type, and that question type differs between the GD and GUI domains. Because these results are inconclusive and inconsistent, there is a need for additional study to understand tool usage for specific question types.

4.5 Participant Feedback

In addition to measuring correctness and efficiency and comparing the results between the DSLs and the GPLs, we wanted to understand whether participants found the DSLs simpler than the GPLs. The accepted view (Bentley 1986) is that it is easier for developers to understand programs written with DSLs than it is to understand programs written with GPLs. We wanted to test this assumption.

After the study, we asked the participants to provide feedback on the simplicity of the language notation (DSL or GPL) and on the complexity of the tasks performed in the study. In Table 14 we show the results relative to notation simplicity. For the FD and GD domains, the participants rated the DSL notation significantly easier to use than the GPL notation. We observed no significant different for the GUI domain.

As a result of careful question preparation, we expected there to be little difference in complexity of the questions between the GPL and the DSL languages. The results in Table 15 support this expectation and show no significant difference between the perceived complexity of the DSL and GPL questionnaires.

Table 14 Simplicity of language notation

		N	Mean	Std. Dev.	Median	Mean rank	Z	p-value*
FD	DSL	27	4.00	0.88	4	32.48	−3.416	**.001**
	GPL	24	2.83	1.20	2.5	18.71		
GD	DSL	27	4.30	0.61	4	34.96	−4.441	**<.001**
	GPL	25	3.20	0.82	3	17.36		
GUI	DSL	20	4.10	0.91	4	21.55	−0.237	.813
	GPL	23	4.22	0.67	4	22.39		

*Mann-Whitney U test

Significant results are highlighted in bold

5 Result Comparison Between Original and Replicated Studies

In this section we compare results from replication to those from the original experiment.

5.1 Participant Domain Experience

In Table 16 we present a comparison of the participants' self-evaluation of their domain experience between the original experiment and the replication. We can make two key observations about this data:

- In all cases, the participants' self-reported domain experience is higher in the replication than in the original study.
- In all three domains, there is no significant difference between the self-reported domain experience for the DSL and the GPL groups.

These two observations help increase the validity of the replication. First, the more experienced replication participants better represent the practitioner population, who would likely have higher domain knowledge. Second, in the original study, in the GUI domain, there was a significant difference between experience of participants who used the DSL and those who used the GPL. Even though this difference was in the opposite direction of the hypotheses (therefore being minor), this threat is not present in the replication.

Table 15 Complexity of the questionnaires

		N	Mean	Std. Dev.	Median	Mean rank	Z	p-value*
FD	DSL	27	2.70	0.95	3	22.44	−1.903	0.057
	GPL	24	3.21	0.98	3	30.00		
GD	DSL	27	3.00	1.47	3	24.76	−0.892	0.372
	GPL	25	3.40	0.82	4	28.38		
GUI	DSL	20	2.95	1.32	3	20.08	−0.968	0.333
	GPL	23	3.30	1.06	4	23.67		

*Mann-Whitney U test

Table 16 Participant experience in original study and replication

		Mean		p-value		Similar?
		Original	Replication	Original	Replication	
FD	DSL	1.48	2.07	0.564	0.953	✓
	GPL	1.52	2.08			
GD	DSL	1.34	2.37	0.414	0.864	✓
	GPL	1.43	2.44			
GUI	DSL	1.36	3.90	< 0.001	0.476	×
	GPL	3.50	4.19			

5.2 Correctness and Efficiency

In Table 17 we show a comparison of the *correctness* results between the original study and the replication. We can make some observations based on this data:

- The correctness scores improved in all six cases from the original study to the replication. It is likely that this increase is due, at least in part, to the use of IDEs in the replication.
- The spread of the data (i.e. the standard deviation) is smaller in the replication than in the original study, suggesting more consistent performance across the participants.
- While the overall correctness scores increased, the relative differences stayed consistent. That is, for all three domains, the participants using the DSL had significantly higher correctness scores.

In Table 18, we show the comparison of the *efficiency* results between the replication and the original study. Again, we can make some observations based on this data:

- The efficiency scores do not have the same type of improvement as seen in the correctness scores. In fact in one case, in the FD DSL questionnaire, participants' efficiency actually decreased. This difference can be explained, at least in part, by the fact that the time portion of the efficiency calculation in the replication included two items not in the original study:
  - The participants in the replication also had to answer questions about which IDE tools they used for each task.
  - Use of IDE tools required additional time.

Table 17 Correctness in original study and replication

		Mean		p-value		Similar?
		Original	Replication	Original	Replication	
FD	DSL	72.79	83.67	< 0.0005	0.002	✓
	GPL	56.50	70.26			
GD	DSL	73.71	85.18	< 0.0005	< 0.001	✓
	GPL	58.84	72.00			
GUI	DSL	64.25	82.04	< 0.0005	0.027	✓
	GPL	43.38	75.10			

Table 18 Efficiency in original study and replication

		Mean		p-value		Similar?
		Original	Replication	Original	Replication	
FD	DSL	1.84	1.46	< 0.0005	< 0.001	✓
	GPL	0.69	0.81			
GD	DSL	1.44	1.65	< 0.0005	< 0.001	✓
	GPL	0.81	0.92			
GUI	DSL	0.94	1.19	< 0.0005	< 0.001	✓
	GPL	0.67	0.98			

- The statistical results between the two studies are consistent, that is, the participants using the DSL are significantly more efficient.

5.3 Individual Question Results

In Table 19 we show a comparison of the original study and the replication relative to question type, i.e. Learn, Understand, Evolve. Based on this data, we can make a few observations:

– For all types of questions in both language types, the correctness increased between the original study and the replication.

Table 19 Correctness of learn, understand, and evolve questions in original study and replication

			Mean		p-value		Similar?
			Original	Replication	Original	Replication	
FD	Learn	DSL	81.37	90.12	0.001	0.000	✓
		GPL	66.20	74.30			
	Understand	DSL	63.68	75.37	0.05	0.085	×
		GPL	54.58	62.92			
	Evolve	DSL	79.41	91.05	<0.0005	0.011	✓
		GPL	50.00	78.47			
GD	Learn	DSL	68.02	75.92	<0.0005	0.222	×
		GPL	51.85	68.00			
	Understand	DSL	74.32	90.00	<0.0005	0.000	✓
		GPL	59.44	74.00			
	Evolve	DSL	78.38	86.42	0.009	0.001	✓
		GPL	64.81	72.67			
GUI	Learn	DSL	57.66	81.66	0.001	0.298	×
		GPL	40.95	75.36			
	Understand	DSL	63.78	76.50	0.001	0.990	×
		GPL	50.86	77.39			
	Evolve	DSL	71.62	91.67	<0.0005	0.000	✓
		GPL	33.33	71.00			

– In most cases, the differences between the language types (DSL and GPL) decreased in the replication.

To better understand this result, we analyzed the individual questions. In Table 20 we show the comparison of the correctness of the questions between the original study and the replication. Based on this data, we can make a few observations:

– For the majority of questions in both the original and the replication, the participants using the DSL performed better than those using the GPL (GPL was better on four questions in the original study and six in the replication).
– The difference in correctness could be due to question complexity. Looking back to Table 8 shows that there is a wide disparity in the correctness for different questions.

The results from the individual questions and the question sets must be taken with caution. For example, the complexity of questions differed across question sets. The results showed that while participants achieved similar correctness on simple tasks in both languages, for complex tasks, the participants achieved higher correctness when using the DSLs. Therefore, because of this variation, we believe that the more reliable conclusions come from the overall results on the full questionnaires.

5.4 Participant Feedback

In Table 21 we show the comparison of the perception of the simplicity of use between the original study and the replication. Based on this data, we can make a few observations:

– In all cases, the replication participants found the DSLs and the GPLs simpler than the original study participants.
– Even with this increased simplicity, the statistical results are consistent between the two studies. The participants viewed the FD and GD DSLs to be significantly simpler than

Table 20 Correctness of individual questions in original study and replication

	FD DSL : GPL		GD DSL : GPL		GUI DSL : GPL	
	Original	Replication	Original	Replication	Original	Replication
Q1	<	=	>	>	>	>
Q2	>	>	>	>	<	<
Q3	>	>	>	>	>	>
Q4	>	>	>	>	>	=
Q5	>	>	<	>	>	>
Q6	>	>	>	>	>	>
Q7	>	>	>	>	>	<
Q8	<	<	>	>	>	<
Q9	>	>	>	>	>	>
Q10	>	<	>	>	>	>
Q11	>	>	>	<	>	>

> better in DSL, < better in GPL, = same (within 1%)

Table 21 Simplicity of use in original study and replication

		Mean		p-value		Similar?
		Original	Replication	Original	Replication	
FD	DSL	3.44	4.00	0.001	0.001	✓
	GPL	2.48	2.83			
GD	DSL	2.97	4.30	0.032	<0.001	✓
	GPL	2.64	3.20			
GUI	DSL	3.50	4.10	0.449	0.813	✓
	GPL	3.36	4.22			

the corresponding GPLs and showed no difference between the simplicity of the GUI DSL and GPL.

In Table 22 we show the comparison of the perception of the complexity of the questionnaires bewteen the original study and the replication. Based on this data, we can make a few observations:

- The complexity ratings for the DSLs either stayed the same or increased, while the complexity ratings for the GPLs all decreased in the replication. Because the GPL programs were longer than the DSL programs, we hypothesize that IDE tools were more helpful for the GPL end-users which resulted in decreased complexity. The participants' perception of complexity in the paper-based questionnaires (the original study) likely was affected by the number of lines of code. The features of the text editors (scrolling, syntax highlighting, etc.) are more helpful for longer programs (in our case, GPL programs).
- The replication study participants found the DSL tasks consistently less complex than the corresponding GPL tasks. This result is opposite the result from the original study. This result could have differed because the original study used a within-subjects design such that each participant worked with both language types. Conversely, the replication used a between-subjects design, so the participants had no basis upon which to calibrate their ratings.

Table 22 Questionnaire complexity in original study and replication

		Mean		p-value		Similar?
		Original	Replication	Original	Replication	
FD	DSL	2.70	3.15	0.196	0.057	✓
	GPL	3.21	2.33			
GD	DSL	3.00	3.00	0.188	0.372	✓
	GPL	3.40	2.89			
GUI	DSL	2.95	3.00	0.545	0.333	✓
	GPL	3.30	2.92			

– The statistical results were consistent between the original study and the replication, there was no significant differences between the DSLs and the GPLs in any of the domain.

6 Threats to Validity

In this section we discuss three type of validity threats: (1) those from the original study that we were not able to address, (2) those from the original study that we were able to address, and (3) new threats introduced in the replication. By making specific changes to the design of the original study to address threats to validity (while introducing different validity threats) and still obtaining similar results, we strengthen the overall conclusion.

6.1 Unaddressed Validity Threats

In Table 23 we list the validity threats from the original study. Because the replication generally followed the lab package from the original study, many of those threats remain. In the next subsection we discuss those threats that we specifically addressed in designing the replication. Here we provide a brief summary of the threats that remain, with the details appearing in our earlier paper.

– It is possible that the specific applications chosen for the study or the complexity of the questions could have affected the results. We have no evidence to suggest that this threat is serious. But, we did not specifically evaluate it.
– We prepared the study materials in English, while the participants were not native English speakers. If the participants had concerns, the experimenters were available to explain the questions to them. While we do not think this threat is significant, we did not study the effects of this choice on the results.

Table 23 Threats to validity in replication

Validity	Issue	Cause	Addressed
Construct	Design threat	The chosen application	no
		The question complexity	no
Internal	Instrumentation	Paper based questionnaires	yes
		Language used in questionnaires	no
		Lack of lessons to GPLs	no
	Selection	Optional participation	no
	Maturation	Tired participants	yes
		Participate in DSL and GPL tests	yes
External	Population	Representative participants	no
		Equal prior experience with domains	yes
	Settings	Representative tasks	no

– In the replication, we again used students as the participants. While we might obtain different results with professionals, the tasks were on a level of complexity that made students the correct participant type. Further replications with professionals will provide additional insights. However, in Slovenia most computer science students in their later years of study have industrial experience, as employment is encouraged by the government. In fact, the study participants likely have more experience than most student participants in other studies. However, this fact does not eliminate the external validity threat, because the participants were not performing the tasks in industrial settings nor were they full-time practitioners, thus this threat remains from the original study.

6.2 Addressed Validity Threats

Part of the goal of the replication was to address some of the validity threats in the original study. In this section, we summarize how we addressed those threats.

– One of the weaknesses of the original study (also indicated by a reviewer of that paper) was that the participants performed all tasks manually on paper. In the replication, we addressed this threat by providing the participants with IDEs to support each task. This change should help to increase the realism of the study tasks.
– To address two validity threats in the original study, we changed from a within-subjects design to a between-subjects design in the replication. This conscious change allowed us to remove the threats related to maturation, that is tired participants and learning between tasks.
– We also removed the threat related to domain experience. In the replication, there were no significant differences.

6.3 New Validity Threats Introduced in the Replication

In the course of making design changes to address threats to validity, we introduced two new threats to validity (Table 24).

– To provide the participants with tool support, we had to choose IDEs for each task. Our choice of IDEs introduces a *construct* threat. Typically the IDEs available to support GPLs are more advanced than those for DSLs. Moreover, for because there was no tool support for the FD DSL, we had to build appropriate DSL tools inside the IDE (e.g. autocomplete, syntax highlighting, and validators) for this domain. Even with these changes, it is possible that the IDEs introduced a threat to the validity of the results.

Table 24 New threats to validity introduced in replication

Validity	Issue	Cause
Construct	Design threat	The chosen IDEs
Construct	Design threat	IDE tools group
Construct	Design threat	Inequivalent IDE tools support
Internal	Selection	Between subject design

- In this replicated study we investigated which tools participants used for each task. Our choice of tool groups (build, editor, find & replace, and other) introduced another threat to *construct validity*. Even though we explained the tool groupings during the training, the participants could have grouped tools differently than we did. While we have no evidence that such a misunderstanding occurred, it is a potential threat.
- By developing IDE tools for the FD domain, we attempted to ensure that GPL users and DSL users had access to similar tools. It is possible that we omitted key features that may be present in the corresponding GPL tool support. This situation raises a potential threat to *construct validity*. Although, since any omitted tools would tend to bias the results towards the GPL, this potential threat, if present, would actually provide more confidence in the results.
- By changing from a within-subjects design to a between-subjects design to address the maturation threat, we introduced an *internal* threat related to selection. Given that there were no significant differences between the experience level of the participants in the different language types, this threat is minimal.

7 Conclusion

In this replicated study, we tackle the shortcomings of the original study by replicating the family of experiments. We modified the original study by having the participants perform the tasks on computers with IDEs rather than on paper. We introduced additional changes to the experimental design (e.g. in the experiment execution and the number of tasks performed by the participants) to obtain more realistic results. The main contributions of this paper are:

- The participants in the replication achieved higher correctness and efficiency than the participants in the original study. The participants' performance improved for both DSL and GPL programs. We believe that the improved results are due, at least in part, to the fact that the participants in the replication used IDEs rather than paper.
- A comparison of the program comprehension results between te original and replicated studies showed that the overall results were similar, that is developers are significantly more accurate and efficient when using DSLs than when using GPLs.
- An in-depth analysis of question sets shows that the results did not hold for all sets of questions as in original study. This result suggest that tool support for program comprehension can reduce the gap in correctness and efficiency.
- While the particular tools favored differed across domains, in all cases, participants used different tools when performing program comprehension tasks on DSLs than they did when performing the same tasks on GPLs.

In this study we confirmed that participants are significantly more effective and efficient with DSLs than with GPLs when using IDEs. While the differences are significant, the absolute difference is relatively small. For example, because the correctness scores for program comprehension using GPLs were high (70% – 75%), the improvement in correctness for DSL was small (82% – 85%). In domains where the participants were less experienced (FD and GD) the improvement in correctness was higher (15% – 16%). In

the domain where the participants had more experience (GUI), the improvement in correctness was lower (7%). While these differences are relatively small, they do suggest that in cases where a DSL is available, developers will perform better when using it compared with using a GPL. Conversely, it may not always be cost-effective to create a new DSL in instances where one does not already exist. Many other factors affect the development of a new DSL including: community size, maintenance costs, and program efficiency.

Based on the results of this replication, we can propose some ideas for DSL researchers and possible directions for empirical studies. Further study is needed to identify which program comprehension tasks are more amenable to DSLs. Another research direction is to explore those DSLs which evolve or extend over time (Erdweg et al. 2012; Mernik 2013). It would be interesting to see empirical results on extended DSL and similar GPL library. The DSLs and corresponding GPL libraries included in our experiments tackle problems that were tied to a single domain. None of DSLs in our study included composite domains, where DSLs are developed using language composition (Reis et al. 2015). It would be interesting to see results of an experiment where participants are familiar with the original domain and use extended or composite features of a DSL or a library in a GPL.

Appendix A: Training Session

To illustrate the type of information the participants saw during training, Fig. 1 provides an example of one of the slides from the DSL GD domain training session. For the sake of clarity in the paper, we simplified the slide from its original use in the training session. The original slide along with the slides for all training sessions for all applications are available in the

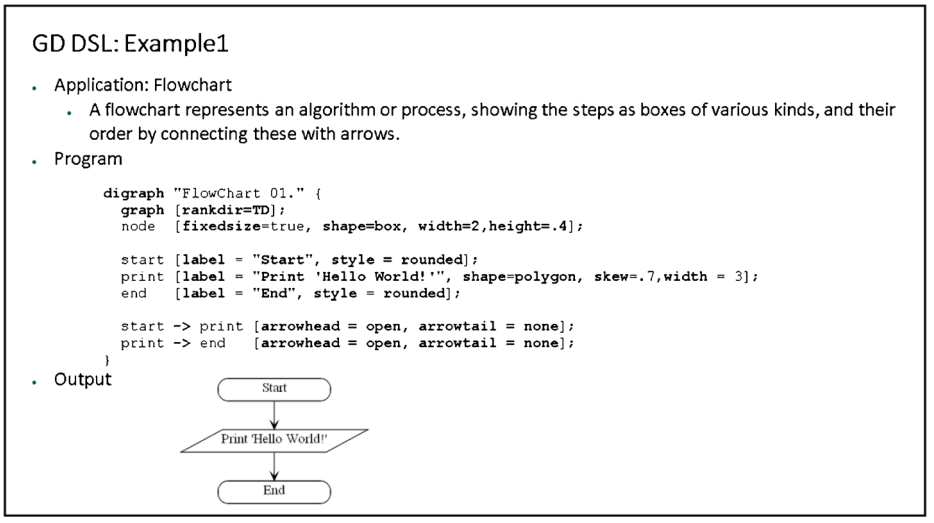


Fig. 1 Slide with DSL program in GD domain used in tutorial session

online lab package[3]. This slide shows example program code (in the middle of the slide) followed by the program output (below the code). During the training session for GD DSL, we explained all language constructs from GD domain (digraph, graph, node, connection) and the individual properties for each construct (for nodes, fixedsize, shape, width, and height).

Appendix B: Code Snippets and Output from each DSL

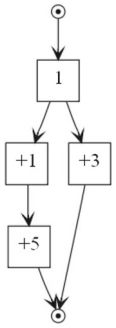
Snippet from FD questionnaire on DSL (question 9)

```
Menu: all ( opt (appetiser), MainCourse,
                Dessert)
MainCourse: one-of( DishA, DishB)
DishA: all ( roastBeef, backedPotatoes)
DishB: all ( stewedSteak, mashedPotatoes)
Dessert: one-of( fruit, iceCream)
```

```
one-of(
all(appetiser, roastBeef , backedPotatoes, fruit)
all(appetiser, stewedSteak, mashedPotatoes, iceCream)
all(appetiser, roastBeef , backedPotatoes, fruit)
all(appetiser, stewedSteak, mashedPotatoes, iceCream)
all(roastBeef  , backedPotatoes, fruit)
all(stewedSteak, mashedPotatoes, iceCream)
all(roastBeef  , backedPotatoes, fruit)
all(stewedSteak, mashedPotatoes, iceCream)
)
```

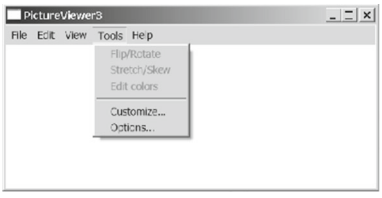
Snippet from GD questionnaire on DSL (question 6)

```
digraph "Branch 6." {
    rankdir=TD;
    ordering = "out";

    start[shape = point, peripheries = 2          ];
    end   [shape = point, peripheries = 2          ];
    "1"   [shape = box   , width = 0.5, height = 0.5];
    "+1"  [shape = box   , width = 0.5, height = 0.5];
    "+3"  [shape = box   , width = 0.5, height = 0.5];
    "+5"  [shape = box   , width = 0.5, height = 0.5];

    start -> "1"  [arrowhead = open, arrowtail = none];
    "1"   -> "+1" [arrowhead = open, arrowtail = none];
    "1"   -> "+3" [arrowhead = open, arrowtail = none];
    "+1"  -> "+5" [arrowhead = open, arrowtail = none];
    "+3"  -> end  [arrowhead = open, arrowtail = none];
    "+5"  -> end  [arrowhead = open, arrowtail = none];

    rank = same; "+1"; "+3";
}
```



Snippet from GUI questionnaire on DSL (question 6)

```
<Window x:Class="WPF_Viewer.Viewer3"
  ...
  Title="PictureViewer3" Height="200" Width="400">
  <DockPanel Name="viewerDockPanel">
    <Border Height="50" BorderThickness="1"
            DockPanel.Dock="Top">
      <StackPanel>
        <Menu IsMainMenu="True"
              DockPanel.Dock="Top" >
          <MenuItem Header="_File"/>
          <MenuItem Header="_Edit"/>
          <MenuItem Header="_View"/>
          <MenuItem Header="_Tools">
            <MenuItem Header="_Flip/Rotate"
                      IsCheckable="False"/>
            <MenuItem Header="_Stretch/Skew"
                      IsCheckable="False"/>
            <MenuItem Header="_Edit colors"
                      IsCheckable="False"/>
            <Separator Height="1"/>
            <MenuItem Header="_Customize..."/>
            <MenuItem Header="_Options..."/>
          </MenuItem>
          <MenuItem Header="_Help"/>
        </Menu>
      </StackPanel>
    </Border>
    <Border Height="30" BorderThickness="1"
            DockPanel.Dock="Bottom"/>
    <Border BorderThickness="1"
            DockPanel.Dock="Right"/>
  </DockPanel>
</Window>
```



[3] https://lpm.feri.um.si/projects/ReplicationLabPackage.zip

References

Albuquerque D, Cafeo B, Garcia A, Barbosa S, Abrahao S, Ribeiro A (2015) Quantifying usability of domain-specific languages: an empirical study on software maintenance. J Syst Softw 101:245–259

Baldassarre MT, Carver J, Dieste O, Juristo N (2014) Replication types: Towards a shared taxonomy. In: Proceedings of the 18th international conference on evaluation and assessment in software engineering. ACM, New York, pp 18:1–18:4

Barišić A, Amaral V, Goulão M (2017) Usability driven DSL development with USE-ME. Computer Languages, Systems & Structures. In: Press

Bentley J (1986) Programming pearls: little languages. Commun ACM 29(8):711–721

Carver JC (2010) Towards reporting guidelines for experimental replications: A proposal. In: 1st international workshop on replication in empirical software engineering

Carver JC, Syriani E, Gray J (2011) Assessing the frequency of empirical evaluation in software modeling research. In: Proceedings of the 1st international workshop on experiences and empirical studies in software modeling (EESSMOD), pp 28–37

Chiş A, Denker M, Gîrba T, Nierstrasz O (2015) Practical domain-specific debuggers using the moldable debugger framework. Comput Lang Syst Struct 44:89–113

Consel C, Marlet R (1998) Architecturing software using a methodology for language development. In: Proceedings of the 10th international symposium on programming language implementation and logic programming, vol 1490, pp 170–194

Cornelissen B, Zaidman A, Van Deursen A (2011) A controlled experiment for program comprehension through trace visualization. IEEE Trans Softw Eng 37(3):341–355

van Deursen A, Klint P (1998) Little languages: little maintenance. J Softw Maint 10(2):75–92

van Deursen A, Klint P (2002) Domain-specific language design requires feature descriptions. J Comput Inf Technol 10(1):1–17

van Deursen A, Klint P, Visser J (2000) Domain-specific languages: an annotated bibliography. ACM SIGPLAN Not 35(6):26–36

Erdweg S, Giarrusso PG, Rendel T (2012) Language composition untangled. In: Proceedings of the twelfth workshop on language descriptions, tools, and applications (LDTA'12). ACM, New York, pp 7:1–7:8

Gansner ER, Koutsofios E, North S (2009) Drawing graphs with dot. Tech. rep., AT&T Bell Laboratories, Murray Hill. http://www.graphviz.org/pdf/dotguide.pdf

Goulão M, Amaral V, Mernik M (2016) Quality in model-driven engineering: a tertiary study. Softw Qual J 24(3):601–633

Häser F, Felderer M, Breu R (2016) Is business domain language support beneficial for creating test case specifications: a controlled experiment. Inf Softw Technol 79:52–62

Hermans F, Pinzger M, Van Deursen A (2009) Domain-specific languages in practice: a user study on the success factors. In: Model driven engineering languages and systems, lecture notes in computer science, vol 5795. Springer, Berlin, pp 423–437

Hevner AR, Linger RC, Collins Webb R, Pleszkoch M, Prowell S, Walton G (2005) The impact of function extraction technology on next-generation software engineering. Tech. Rep CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University

Hoisl B, Sobernig S, Strembeck M (2014) Comparing three notations for defining scenario-based model tests: a controlled experiment. In: Proceedings of the 9th international conference on the quality of information and communications technology (QUATIC), pp 95–104

Hudak P (1998) Modular domain specific languages and tools. In: Proceedings of the 5th international conference on software reuse (JCSR '98), IEEE Computer Society, pp 134–142

Johanson AN, Hasselbring W (2016) Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment. Empir Softw Eng 22(4):2206–2236

Jung HW, Kim SG, Chung CS (2004) Measuring software product quality: a survey of iso/iec 9126. IEEE Softw 21(5):88–92

Kitchenham B (2008) The role of replications in empirical software engineering—a word of warning. Empir Softw Eng 13(2):219–221

Kosar T, Oliveira N, Mernik M, Varanda Pereira MJ, Črepinšek M, da Cruz D, Henriques PR (2010) Comparing general-purpose and domain-specific languages: an empirical study . Computer Science and Information Systems 7(2):247–264

Kosar T, Mernik M, Carver J (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

MacVittie LA (2006) XAML in a nutshell. O'Reilly Media, Inc, Sebastopol
Mauw S, Wiersma W, Willemse T (2004) Language-driven system design. Int J Softw Eng Knowl Eng 6(14):625–664
Meliá S, Cachero C, Hermida JM, Aparicio E (2016) Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study. Softw Qual J 24(3):709–735
Mernik M (2013) An object-oriented approach to language compositions for software language engineering. J Syst Softw 86(9):2451–2464
Mernik M, Heering J, Sloane A (2005) When and how to develop domain-specific languages. ACM Comput Surv 37(4):316–344
Nugroho A (2009) Level of detail in UML models and its impact on model comprehension: a controlled experiment. Inf Softw Technol 51(12):1670–1685
Prähofer H, Schatz R, Wirth C, Hurnaus D, Mössenböck H (2013) Monaco – a domain-specific language solution for reactive process control programming with hierarchical components. Comput Lang Syst Struct 39(3):67–94
Reis LV, Di Iorio VO, Bigonha RS (2015) An on-the-fly grammar modification mechanism for composing and defining extensible languages. Comput Lang Syst Struct 42:46–59
Shull FJ, Carver JC, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empir Softw Eng 13(2):211–218
Sprinkle J, Mernik M, Tolvanen JP, Spinellis D (2009) Guest editors introduction: what kinds of nails need a domain-specific hammer? IEEE Softw 26(4):15–18
Storey MA (2005) Theories, methods and tools in program comprehension: past, present and future. In: Proceedings of the 13th international workshop on program comprehension (IWPC'05), IEEE Computer Society, pp 181–191
Umuhoza E, Brambilla M, Ripamonti D, Cabot J (2015) An empirical study on simplification of business process modeling languages. In: Proceedings of the 2015 ACM SIGPLAN international conference on software language engineering. ACM, New York, pp 13-24
Varanda Pereira MJ, Fonseca J, Henriques PR (2016) Ontological approach for DSL development. Comput Lang Syst Struct 45:35–52
Wile DS (2001) Supporting the DSL spectrum. J Comput Inf Technol 9(4):263–287
Williams M (2002) Microsoft visual c# (core reference). Microsoft Press, Redmond

Tomaž Kosar received the Ph.D. degree in computer science at the University of Maribor, Slovenia in 2007. His research is mainly concerned with design and implementation of domain-specific languages. Other research interest in computer science include also domain-specific modelling languages, empirical software engineering, generative programming, compiler construction, object-oriented programming, object-oriented design, refactoring, and unit testing. He is currently an Assistant Professor at the University of Maribor, Faculty of Electrical Engineering and Computer Science.

Sašo Gaberc received, from University of Maribor, Faculty of Electrical Engineering and Computer Science, a B.Sc. in Computer Science and Information Technologies (2012) and a M.Sc. in Computer Science and Information Technologies (2015). He is currently working as a full time software developer in industry.

Jeffrey C. Carver received the PhD degree in computer science from the University of Maryland in 2003. He is a full professor in the Department of Computer Science, University of Alabama. His main research interests include empirical software engineering, peer code review, human factors in software engineering, software quality, software engineering for science, and software process improvement. He is a Senior Member of the IEEE Computer Society and the ACM.

Contact him at carver@cs.ua.edu.

Marjan Mernik received the M.Sc. and Ph.D. degrees in computer science from the University of Maribor in 1994 and 1998 respectively. He is currently a professor at the University of Maribor, Faculty of Electrical Engineering and Computer Science. He is also a visiting professor at the University of Alabama at Birmingham, Department of Computer and Information Sciences, and at the University of Novi Sad, Faculty of Technical Sciences, Serbia. His research interests include programming languages, compilers, domain-specific (modeling) languages, grammar-based systems, grammatical inference, and evolutionary computations. He is a member of the IEEE, ACM and EAPLS. Dr. Mernik is the Editor-In-Chief of Computer Languages, Systems and Structures journal, as well as Associate Editor of Applied Soft Computing journal.

He is being named a 2017 Highly Cited Researcher.