Influence of domain-specific notation to program understanding

Tomaž Kosar*, Marjan Mernik*, Matej Črepinšek*, Pedro Rangel Henriques†,
Daniela da Cruz†, Maria João Varanda Pereira‡ and Nuno Oliveira†

* University of Maribor, Faculty of Electrical Engineering and Computer Science,
Smetanova 17, 2000 Maribor, Slovenia
Email: {tomaz.kosar, marjan.mernik, matej.crepinsek}@uni-mb.si
† University of Minho - Department of Computer Science,
Campus de Gualtar, 4715-057, Braga, Portugal
Email: {prh, danieladacruz, nunooliveira}@di.uminho.pt
‡ Polytechnic Institute of Bragança
Campus de Sta. Apolónia, Apartado 134 - 5301-857, Bragança, Portugal
Email: mjoao@ipb.pt

Abstract—**Application libraries are the most commonly used implementation approach to solve problems in general-purpose languages. Their competitors are domain-specific languages, which can provide notation close to the problem domain. We carried out an empirical study on comparing domain-specific languages and application libraries regarding program understanding. In this paper, one case study is presented. Over 3000 lines of code were studied and more than 86 pages long questionnaires were answered by end-users, answering questions on learning, perceiving and evolving programs written in domain-specific language as well as general-purpose language using application library. In this paper, we present comparison results on end-users' correctness and consumed time. For domain-specific language and application library same problem domain has been used – a well-known open source graph description language, DOT.**

I. Introduction

A domain-specific language (DSL) is a language constructed to provide a notation close to an application domain, and is based only on the concepts and features of that domain [9]. As such, a DSL is a means of describing and generating members of a program family within a given problem domain, without the need for knowledge about general programming. By providing notations close to the application domain, a DSL offers many advantages. On the other hand, in combination with an application library, any general-purpose language (GPL) can act as a DSL. Furthermore, GPLs are the most commonly used method to solving programming problems.

However, DSLs have in productivity numerous advantages over GPLs – they are more expressive for the domain in question, with corresponding gains in productivity and reduced maintenance costs [17]. Some specific goals of DSLs such as:

- to make programming more accessible to end-users,
- to improve correctness of the written programs, and
- to improve the program developing time

seems to follow implicitly from the DSL definition. But, were these claims really proved in practice? All the above claims have a common denominator in the assertion that DSL programs are easier to comprehend than GPL programs.

Program Comprehension (PC) [2], [14] is a hard cognitive task. This is usually done by the construction of a mental model of the program, trying to conceive the meaning of that model [15]. One of the objectives in our project[1] is to measure how easier is to comprehend programs written in DSLs than GPLs. Specifically, in this paper we try to bring confirmation of the hypothesis that *DSLs are easier to understand than GPLs*. This hypothesis is defined from the literature by the experiment under controlled programming environment, using direct observation, and questionnaires to measure the end-user understanding of DSL and GPL programs. Both questionnaires are on solving the same problem domain.

The organization of this paper is as follows. Related work on domain-specific languages and program comprehension is discussed in Section II. The definition of the precise focus of our experimental study, the identification of its main goals, and the choice of a skeleton for it, according to the goals, are the topics introduced in Section III. Details about the experiment carried out are given in Section IV. Key findings are given in Section V and concluding remarks with future work are summarized in Section VI.

II. Related work

Empirical research in software engineering is a difficult but important discipline. In order to avoid questionable results from research, certain conditions must be considered while preparing an experiment. This work follows the framework [1] introduced to motivate and replicate studies. The proposed framework concentrates on building knowledge about the context of an experiment and is based on organizing sets of related studies (family of studies). These studies contribute

[1]This work is sponsored by bilateral project "Program Comprehension for Domain-Specific Languages" (code BI-PT/08-09-008) between Slovenia and Portugal.

to common hypotheses which does not vary for individual experiments. Therefore, the following guidelines from [1] have been followed in order to prepare this experiment:

- context of the study,
- comparison validity,
- measurement framework, and
- presentation of key findings, focused on hard/unexpected/controversial results.

The above listing is further explained in Sections III and IV.

Before this study, authors of this experiment were involved in the another experiment where they obtained adequate knowledge in the preparation of experiments in field of software engineering. Various implementation approaches for developing a DSL exists and in the paper [8] experiment on ten diverse implementation approaches for DSLs has been conducted using the same representative language FDL (Feature Description Language) [4]. The experiment shows, that approaches differ in terms of the effort needed to implement them. Also, the paper provides empirical comparison on end-users effort to write programs using the various DSL implementation approaches. Beside that, the paper presents empirical comparison on DSL programs and equivalent GPL programs, measured with effective lines of code. This part of the experiment is similar to this paper, however the focus in the paper [8] is more on productivity, while here is on end-users program understanding. Also, the problem domain in the experiment [8] is different (feature descriptions) than in this paper (graph descriptions).

The another experiment in field of software engineering can be found in a work carried out by Prechelt [12], which provides the comparison of seven programming languages (C, C++, Java, Perl, Python, Rexx, and Tcl). The advantages and disadvantages of those programming languages are discussed on a single given problem (string processing program). Acquired programs were compared on run times, memory consumption, source text length, comment density, program structure, reliability, and the amount of effort required to write them.

In this paper, we do not want to discuss the construction of program comprehension tools for DSLs, neither measure the efficiency of these tools. We just want to infer from empirical study how easy is to understand a DSL comparing to a GPL. For a DSL program understanding, the study of cognitive models can give an important contribute to our work [14]. They are related with the way we organize the information and the strategies used to understand programs and systems [16].

III. Experiment goals and contour

Before preparing the experiment, the first concern was the goal of the experiment. To define that, we had to clarify the difference between the program comprehension and the program understanding.

Studying literature, we came to the distinction that usually, the program comprehension is related with real applications development and maintaining, while the program understanding is related with the program analysis. Although, the aim of the experiment was to measure the programmers effort to understand DSL and GPL programs, we decided not to include into the study the influence of the development environment. Development is strongly connected with tools, and both GPL and DSL development environments provide different tools. Hence, comparing DSL and GPL tools is outside of this research interests.

The next question was, how to measure the program understanding. There are several possibilities. In this experiment it has been decided to resort to questionnaires measuring end-users understanding capabilities, assessing their performance when analysing/interpreting programs. Two questionnaires have been prepared for the program understanding of DSL and GPL programs.

Then, the structure of questionnaires has been defined according to the hypothesis of the work, that DSL programs are easier to understand than GPL programs. Now, more specific claims about program understanding have been defined that DSL programs are easier to *learn*, *perceive*, and *evolve* than GPL programs. Questions have been prepared to cover this three groups: learn, perceive, and evolve. In the first group, questions on learning notation and meaning of programs have been given to the end-users. In the second group, questions on program perceiving have been defined, such as identification of: correct meaning from the given program, language constructs, new construct meaning, and meaning of a program with given comments. In the third group, end-users had been challenged to expand/remove/replace program functionality.

For these three groups, 11 questions have been defined:

- Learn
  - Q1 Select syntactically correct statements.
  - Q2 Select program statements with no sense (unreasonable).
  - Q3 Select valid program with the given result.
- Perceive
  - Q4 Select the correct result for the given program.
  - Q5 Identify language constructs.
  - Q6 Select program with the same result.
  - Q7 Select the correct meaning for the new language construct.
  - Q8 Identify language constructs in the program with comments.
- Evolve
  - Q9 Expand the program with new functionality.
  - Q10 Remove functionality from the program.
  - Q11 Change functionality from the program.

Both, DSL and GPL questionnaires have been constructed using the above questions.

To illustrate the style of the questions used in the questionnaires, an example is presented in Figure 1. This example is a fragment of concrete Q4 (see listing above) for the DOT language considered in this experiment – end-users are given the DOT programs and they have to find a correct meaning. Figure 1 shows only the correct choice (without false ones). In all learn and perceive questions the end-users have to select

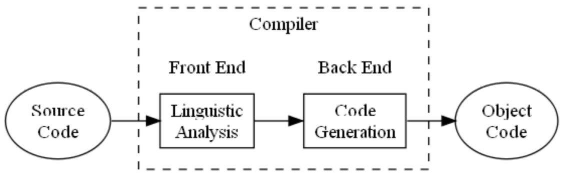
Question 8

QC012 DOT-CompilerConstruction: Please select valid figure of the following DOT program:

Choose one answer.

```
digraph "Figure 4." {
  graph [rankdir=LR];
  subgraph cluster_0 {
    graph [label=Compiler,  style=dashed];
    subgraph cluster_0_1 {
      graph [color=white, label="Front End"];
      "Linguistic\nAnalysis"  [shape=box];
    }
    subgraph cluster_0_2 {
      graph [color=white, label="Back End" ];
      "Code\nGeneration"  [shape=box];
    }
  }
  "Source\nCode"          -> "Linguistic\nAnalysis";
  "Linguistic\nAnalysis"  -> "Code\nGeneration";
  "Code\nGeneration"      -> "Object\nCode";
}
```

b.



Question 8

QC012 DOT-Flowchart: Please select valid figure of the following DOT program:

Choose one answer.

```
void createGraph4(Agraph_t *g) {
    agnodeattr(g, "fixedsize", "true");
    agnodeattr(g, "width"    , "2"   );
    agnodeattr(g, "height"   , "0.4" );

    agedgeattr(g, "arrowhead", "open");

    Agnode_t *start = agnode(g, "Start");
    agsafeset(start, "shape"    , "box"   , "");
    agsafeset(start, "style"    , "rounded","");

    Agnode_t *print_text = agnode(g, "Print 'Input
                         rectangle length\n(l)
                         and width (w)'"   );
    agsafeset(print_text, "shape"    , "parallelogram","");
    agsafeset(print_text, "width"    , "3"     , "");
    agsafeset(print_text, "height"   , "0.8"   , "");

    Agedge_t *start_reada = agedge (g, start, print_text);

    Agnode_t *read_l = agnode(g, "Read l");
    agsafeset(read_l, "shape"    , "parallelogram","");

    Agedge_t *print_readl = agedge (g, print_text, read_l);

    Agnode_t *read_w = agnode(g, "Read w");
    agsafeset(read_w, "shape"    , "parallelogram","");

    Agedge_t *readl_readw = agedge (g, read_l, read_w);

    Agnode_t *calc = agnode(g, "a = l * w");
    agsafeset(calc, "shape"    , "box" , "");

    Agedge_t *readw_calc = agedge (g, read_w, calc);
    Agnode_t *print_a = agnode(g, "Print 'a2'");
    agsafeset(print_a, "shape"    , "parallelogram","");
    Agedge_t *calc_print = agedge (g, calc, print_a);
    Agnode_t *end = agnode(g, "End"   );
    agsafeset(end, "shape"    , "box"   , "");
    agsafeset(end, "style"    , "rounded","");
    Agedge_t *print_end = agedge (g, print_a, end);
}
```

b.

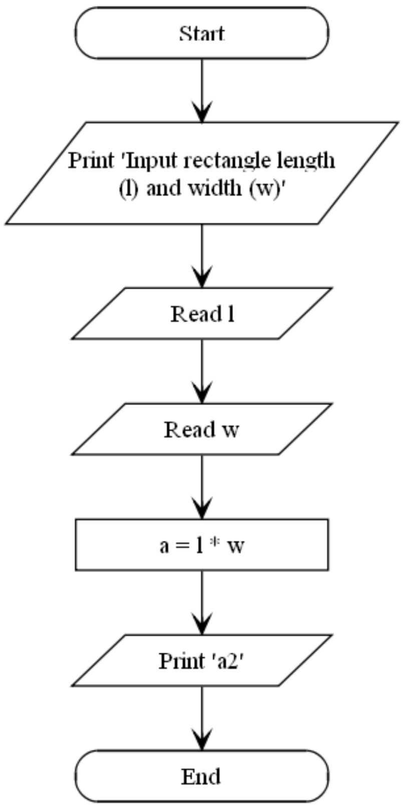


Fig. 1. The question 8 in DSL and GPL questionnaires with the correct choice

correct answers among five given choices. In the questions group evolve, to the end-users are given program listings and they have to remove/expand/replace program parts. The complete questionnaires can be found at the project group webpage[2].

IV. Experiment details

Many problems have to be faced while preparing the experiment. In the following subsections these issues are exposed and discussed according to their influence on comparison validity.

[2]http://epl.di.uminho.pt/~gepl/DSL/

A. Subject of comparison

Over 3000 programming languages, general-purpose as well as domain-specific, have been developed in the past [10], [11]. It is unreal to expect, that complete comparison of domain-specific and general-purpose languages will ever be done. However, empirical research in software engineering is an important step [1], [3], [13]. This kind of experiments are step towards this goal.

There are many different DSLs (focused on different targets and following different implementation approaches [8], [9]). DSLs can have a more procedural (imperative) style or follow a more declarative one. In the procedural case, those languages describe data and operations over data. The declarative DSLs usually describe high level specifications, data or activity models. This can also be true for application libraries. In combination with application library, any GPL can act as a DSL. Domain-specific functionality in application libraries is simply achieved through an object creation and a method invocation (if object-oriented GPL is used). Therefore, DSLs can be compared and related with GPLs. Specifically, this experiment aims to provide some empirical information for comparing end-user understanding on DSL and GPL programs.

In order to prepare the experiment to obtain realistic results, it has been decided to test DSL and GPL programs on the same domain. In order to choose our case study domain, we took into account the fact that we need a domain that is implemented in both notations – domain-specific and application library. One good language that we had been using for graph descriptions is the DOT language [5]. With the DOT language, a directed and an undirected graph can be defined. Graphs show nodes and relations (edges) between them. Beside those constructs, a subgraph inside a graph can be defined. Associated with graphs, nodes edges and subgraphs various attributes can be applied. These attributes control properties such as color, shape, and line styles. The DOT language does not support rendering, viewing, and manipulation of graphs. There are several programs that can do that (GraphViz [6]).

The DOT can be used as a plain text language or application library in the C programming language. This ability makes the DOT perfectly suitable for this experiment.

B. *Conditions when applying comparison*

The results from an experiment can be spread out when repetition of experiment can be proven [13]. Repetition is strongly connected to agreements set down before starting the experiment [1]. Consistency of results in our experiment was obtained creating appropriate conditions to the end-users: using well-structured questionnaires, domain tutorials and extra explanations in their native language.

Before starting the experiment, the following steps have been taken:

- a short tutorial to the end-users has been given on the problem domain,
- a tutorial on domain specific notation together with an example of a program has been given to the end-users,
- a tutorial on application library together with an example of a program has been given to the end-users,
- a tutorial has been given to the end-users in their native language, but the slides, programs and experiment questionnaires were in English,
- the slides and the examples has been given to end-users and could be used during the experiment,
- the first version of questionnaires has been given to a small group as a training set in order to get feedback,
- a feedback from a training set has been used for refining the questions before applying them in the experiment.

TABLE I
END-USER EXPERIENCES IN PROGRAMMING (N = 29)

	Average	Median	St.dev.
Skills in programming	**3.80**	**4**	**0.77**
Skills of programming in C	**3.45**	**3**	**0.83**
Prior experience with DSLs	**2.69**	**2**	**1.11**

N = number of received questionnaires

The following conditions have been defined about the end-users of the questionnaires:

- the end-users must have some experience in GPLs, and
- the DSL experience can be none or poor.

In Table I self-evaluation of the end-users knowledge in programming, programming in the C language and prior experiences with DSLs can be observed. Results given for the latter one significantly differs from their programming experiences in GPLs. A five-graded scale, going from very bad (1) to very good (5) was used for self-evaluation questionnaires (in Tables I, II, III and V). Note, that the column "Average" (in Table I) shows the average value given by 29 end-users, "Median" stands for the middle value in set of end-user grades (number of grades above and below median is exactly the same) and "St. Dev." represents standard deviation on given opinions.

Additionally, we define the following rules for the questionnaire implementors:

- the same group of questions for both experiment on a GPL and a DSL (see Section III) must be used,
- the questions for two applications on the same question groups were prepared (easier and harder application domain),
- the learning and perceiving questions must be multiple choice question,
- the evolving questions must be essay question (end-users are challenged to add code to existing one),
- the questions on four different applications must be defined bya single individual to obtain the same level of question complexity, and
- the questions and the given choices (programs) must be reviewed by other domain experts, to obtain a code as optimal as possible.

C. *Experiment validity*

Reliability of results is hardly connected to the experimental environment and presenting empirical results without explaining environment details is risky [1]. To restrict the impact of the environment on the experiment results, following threats to the validity are given below.

Chosen domain

In Table II, end-users familiarity with the graph description domain is presented, together with experience on the DOT language and library application in C. It can be observed that end-users had almost none experience with the graph description domain, language or application library. From that point of view, the end-users had no advantage in knowledge of

TABLE II
END-USER KNOWLEDGE IN GRAPH DESCRIPTIONS DOMAIN AND ITS IMPLEMENTATIONS (N = 29)

	Average	Median	St.dev.
Familiarity with graph descriptions domain	**1,38**	**1**	**0.82**
Knowledge of DOT language	**1.34**	**1**	**0.77**
Knowledge of DOT application library	**1.38**	**1**	**0.92**

TABLE III
END-USER KNOWLEDGE IN APPLICATION DOMAINS (N = 29)

	DOT application	Average	Median	St.dev.
DSL	**Compiler construction**	**2.86**	**3**	**1.10**
	Branching game	**2.14**	**2**	**1.25**
GPL	**UML**	**3.14**	**3**	**1.12**
	Flowchart	**3.03**	**3**	**1.12**

any implementation and with that, this is not a serious threat to validity of this experiment.

The another concern on the results is how would better knowledge of the problem domain effect on comparison results. As we can see from Table II, the end-users were completely unfamiliar with the DOT domain (median value is 1). To advocate results of the DOT domain, family of experiments with the other problem domain needs to be done and some well-known domains have to be included in the study.

DOT applications

In Table III, the DOT applications and the end-users familiarity with them before starting the experiment is presented. As we can see from the results in Table III, the end-users had average knowledge on compiler construction, UML notation and flow charts (median value 3) and less experience in application of branching game (median value 2). It can be concluded, that the end-user knowledge on the DOT applications used in the GPL questionnaire (UML and flowcharts) were better then knowledge on the DSL questionnaire (compiler construction and branching game). This fact could have minor influence on comparison results – slightly worse results on DSL questions are expected than, if knowledge on DSL DOT applications would be equal to GPL DOT applications.

End-user experience

In the experiment, experienced students from third, fourth and fifth year of undergraduate computer science studies were included. Students came with different background and knowledge, and could have influence on results and an effect on repetition of the experiment [3]. In Table I, we presented results from self evaluation test, where students grade their general knowledge about programming, programming in the C language and prior experience with DSLs. As it has been stated above, one of the concerns on the experiment is how would greater experience with DSLs affect the results on comparison with GPLs. However, it is hard to find a representative sample of end-users for the purpose of the experiment where they would have equal prior experience in using DSLs and GPLs.

Comparability of DSL with GPL questionnaires

Same type of questions in the DSL and the GPL questionnaires were reviewed and calculated by number of graphical elements (nodes, relations and sub-graphs), to obtain the same level of complexity. In Table IV number of graphical elements are presented for both questionnaires (number is calculated on the given program or on the correct choice if question does not contain any program). Observing the same question in the GPL and the DSL questionnaires shows, that programs in DSL questions are at least comparable by the number of elements to same type of GPL questions (often the number of elements is bigger).

However, comparison of questions based on the number of graphical elements has to be taken with caution. Elements of the DOT language are not equivalent in complexity – for instance, definition of subgraphs are much more demanding to understand then definition of nodes (see Figure 1). Therefore, questions have been studied again and redefined (remove/add some of the graphical elements) if non-equivalence on questions complexity has been discovered. Table IV shows the comparison on graphical components after redefinition.

Order of experiments

Both, DSL and GPL, questionnaires were done by two groups of the students. All students conducted both experiments – DSL and GPL questionnaires. However, the order of experiments was different with both groups. To eliminate effect of questionnaires order on results, the first group started with answering questionnaire on DSL and the second with GPL questionnaire (and then proceeded on GPL/DSL questionnaire).

V. RESULTS

In this section we present results on comparison between DSL and GPL program understanding. Comparison shows empirical data on the end-user questions success. Additionally, comparison on the end-user effort in terms of program length and used time to complete the questionnaires is presented.

A. End-users opinion on notations

After finishing both questionnaires the end-users were advised to evaluate both DSL and C implementations. The results on the end-user notation opinion is presented in Table V. Results show, that the end-users liked a DSL more than application library in the language C, however much bigger difference between a DSL and a GPL was expected. In general, the end-users opinion was, that they were familiar to the programming language C notation, while the DOT language notation was first seen in this experiment. It can be concluded, that this result is strongly connected to the threat of the experiment validity shown in Table I – bigger difference would be obtained if experiences in both, the DSL and the GPL notation, were equal for the end-users.

TABLE IV
GRAPHICAL COMPONENT COMPARISON

	DSL			GPL		
	Compilers	**Branching game**	**Average**	**UML**	**Flowcharts**	**Average**
Question 1	**5**	**7**	**6.0**	**3**	**7**	**5.0**
Question 2	**5**	**10**	**7.5**	**3**	**9**	**6.0**
Question 3	**5**	**12**	**8,5**	**5**	**9**	**7.0**
Question 4	**12**	**10**	**11.0**	**5**	**13**	**9.0**
Question 5	**16**	**9**	**12.5**	**3**	**14**	**8.5**
Question 6	**13**	**17**	**15.0**	**3**	**19**	**11.0**
Question 7	**9**	**10**	**9.5**	**4**	**10**	**7.0**
Question 8	**13**	**10**	**11.5**	**13**	**16**	**14.5**
Question 9	**14**	**7**	**10.5**	**15**	**13**	**14.0**
Question 10	**15**	**24**	**19.5**	**16**	**16**	**16.0**
Question 11	**17**	**24**	**20.5**	**13**	**20**	**16.5**

TABLE V
END-USER JUDGMENT ON NOTATIONS AFTER EXPERIMENT (N = 29)

		Average	**Median**	**St.dev.**
DSL	**DOT language notation**	**2.97**	**3**	**0.78**
GPL	**DOT API design in C language**	**2.55**	**3**	**0.68**

B. End-users success on questionnaires

All together, the end-users answered 11 types of questions (with two applications) on both experiments (Table VI). It can be observed that in the most questions, our expectations were correct about the easier and harder application domain. For instance, the end-user success in the DSL questionnaire was better for the compiler construction domain (application 1 in DSL) than results on the branching game (application 2 in DSL) – see the end-users success on questions from 3 to 11. This is also confirmed by results on the end-users opinion about knowledge on both application domains (see Table III). Prior knowledge on GPL applications of UML (application 1 in GPL) and flowchart (application 2 in GPL) were quite equal (see Table III) and that this resulted in questionnaire success rate. Flowchart questions, which were expected to be harder, gave better results in some questions than questions on UML (questions 2, 5, 7, 8, 9 and 10), although the number of components in flowchart programs was often much bigger than for UML programs (see Table IV).

The average value on both applications has been calculated for both DSL and GPL questions. In question 1, the end-users knowledge on DSL and GPL syntax on DOT has been tested. In questions ("Please select correct DOT statements (without syntax errors)") average success rate on DSL was 83,78% and equivalent question in GPL 69,44% – the difference on using DSL instead of GPL in terms of learning new notation was 14,34%. Also, in question 2 ("Please select DOT program with no sense (unreasonable - incorrect compiler diagram)" for the first application on DSL) much better results were obtained by DSL notation then with GPL. It turned out, that question 2 was hard for the end-users (note, poor results in all DOT applications). In question 2 all given programs were correct (executable), however one program produced a semantically incorrect diagram.

Looking at the average success on questions, in most cases the end-users were much more successful in DSL questions then in equivalent GPL questions. Except of question 5. From the results it can be observed that average success rate on the first application in both questionnaires (compiler construction and UML) gave better results on DSL (94,59%) then on GPL (63,89%). However, results on the second domain were unexpected – much better results were obtained by GPL question (success rate 51,35% on DSL and 88,89% on GPL question). Having closer look on both question uncovers that the DSL question was much more complex than the GPL.

Very close results were also obtained by questions 8. Question 8 is connected with measuring effect of comments on the end-users program understanding. It can be concluded that DSLs already contain only relevant concepts from the domain and the usefulness of comments on the end-user understanding of domain-specific programs is rather small. On the other hand, comments in GPL programs bring useful information to the end-user while they often contain descriptions of program close to domain concepts and with that they are helpful for the end-user understanding.

Making general conclusions on basis of the average value of two questions can be extremely risky. That can be learned from the results on question 5, as described above. Although paying attention on different experiment variables (equivalent number of graphical elements, experts overview of questions, giving tests to the training group, etc), still, some questions can be a serious threat to validity of the experiment results. Therefore, more questions have to be defined for the individual hypothesis to get more reliable results. We did that by grouping questions for the hypothesis (on learn, perceive and evolve) as explained in Section IV. In Table VII the average success rate on questions by the individual group are presented. Table VII confirms our hypothesis that the program understanding in terms of learn, perceive and evolve is much better for domain-specific programs than on general-purpose programs.

TABLE VI
COMPARISON OF END-USER SUCCESS ON QUESTIONS (N = 38)

	DSL			GPL			
	Compilers	**Branching game**	**Average**	**UML**	**Flowchart**	**Average**	**Dif.**
Q1	**72,97%**	**94,59%**	**83,78%**	**91,67%**	**47,22%**	**69,44%**	**14,34%**
Q2	**35,14%**	**54,05%**	**44,59%**	**16,67%**	**25,00%**	**20,83%**	**23,76%**
Q3	**78,38%**	**72,97%**	**75,68%**	**80,56%**	**50,00%**	**65,28%**	**10,40%**
Q4	**97,30%**	**78,38%**	**87,84%**	**91,67%**	**58,33%**	**75,00%**	**12,84%**
Q5	**94,59%**	**51,35%**	**72,97%**	**63,89%**	**88,89%**	**76,39%**	**-3,42%**
Q6	**67,57%**	**56,76%**	**62,16%**	**38,89%**	**27,78%**	**33,33%**	**28,83%**
Q7	**89,19%**	**81,08%**	**85,14%**	**41,67%**	**69,44%**	**55,56%**	**29,58%**
Q8	**72,97%**	**54,05%**	**63,51%**	**50,00%**	**63,89%**	**56,94%**	**6,57%**
Q9	**75,68%**	**70,27%**	**72,97%**	**41,67%**	**72,22%**	**56,94%**	**16,03%**
Q10	**94,59%**	**86,49%**	**90,54%**	**66,67%**	**72,22%**	**69,44%**	**21,10%**
Q11	**91,89%**	**81,08%**	**86,49%**	**83,33%**	**52,78%**	**68,06%**	**18,43%**

TABLE VII
AVERAGE END-USER SUCCESS ON LEARN, PERCEIVE AND EVOLVE (N = 38)

	Question	**DSL**	**GPL**	**Difference**
Learn	**Q1, Q2, and Q3**	**68,02%**	**51,85%**	**16,17%**
Perceive	**Q4, Q5, Q6, Q7, and Q8**	**74.32%**	**59.44%**	**14.88%**
Evolve	**Q9, Q10, and Q11**	**83.33%**	**64.81%**	**18.52%**

C. Program length

Another comparison is done in terms of the number of lines of code (LOC). We measured the size of code with the numbers of effective lines of code (*eLOC*) – that is, all lines that are not blanks, standalone-braces or parentheses. In Table VIII we present eLOC for correct question choices from questionnaires. Comparing eLOC for DSL and GPL programs on the same type of question, reveals that GPL programs are much bigger then those in DSLs. This comparison has to be taken with caution and must include number of components from Table IV into account.

Comparing program length presented in Table VIII seems unfair, since we compare length of different programs (for instance, in question 1 DSL program length on compiler is compared with GPL program length on UML).

Therefore, we prepare programs of both GPL application domains (UML and flowchart) also in DSL. Results are presented in Table IX and for instance, the program in question 7 for the first domain (UML program) has 7 eLOC in the domain-specific notation and equal program in the C language 22 eLOC.

Note, that in the general-purpose notation only eLOC for defining graph, nodes, edges and attributes were measured. To that number we did not accumulate lines needed to render graph, preparation of graph layout (DOT), writing diagram to console/file, etc. So, approx. 20 eLOC were eliminated from Tables VIII and IX on GPL programs.

Tables VIII and IX confirm that GPL programs are much bigger in size than DSL programs. Also, from Table IX, it can be observed that when applications in the domain increase, the ratio between length of DSL and GPL programs stays the same (compare questions 4 and 10 for application on UML).

TABLE VIII
PROGRAM LENGTH IN QUESTIONNAIRES

	DSL		GPL	
	Compilers	**Branching game**	**UML**	**Flowchart**
Question 1	**3**	**8**	**11**	**31**
Question 2	**6**	**14**	**6**	**32**
Question 3	**7**	**16**	**27**	**19**
Question 4	**16**	**13**	**32**	**30**
Question 5	**21**	**7**	**18**	**31**
Question 6	**17**	**18**	**23**	**43**
Question 7	**5**	**14**	**22**	**47**
Question 8	**14**	**18**	**24**	**38**
Question 9	**15**	**13**	**104**	**29**
Question 10	**20**	**22**	**102**	**40**
Question 11	**23**	**22**	**102**	**69**

TABLE IX
PROGRAM LENGTH FOR SAME DOMAINS (UML AND FLOWCHART) IN DSL AND GPL

	DSL		GPL	
	UML	**Flowchart**	**UML**	**Flowchart**
Question 1	**6**	**11**	**11**	**31**
Question 2	**6**	**12**	**6**	**32**
Question 3	**8**	**13**	**27**	**19**
Question 4	**11**	**20**	**32**	**30**
Question 5	**5**	**22**	**18**	**31**
Question 6	**6**	**23**	**23**	**43**
Question 7	**7**	**19**	**22**	**47**
Question 8	**14**	**17**	**24**	**38**
Question 9	**35**	**15**	**104**	**29**
Question 10	**34**	**18**	**102**	**40**
Question 11	**34**	**29**	**102**	**69**

TABLE X
AVERAGE END-USER TIME EFFORT ON LEARN, PERCEIVE AND EVOLVE QUESTION GROUPS (N = 38)

	DSL	GPL	Difference
Learn	**18 min 58 sec**	**29 min 48 sec**	**57.09%**
Perceive	**22 min 49 sec**	**29 min 27 sec**	**29.08%**
Evolve	**12 min 36 sec**	**18 min 08 sec**	**44.02%**
Total	**54 min 23 sec**	**1h 17 min 23 sec**	**44.02%**

D. End-user time effort

Before starting the experiment, it has been decided to measure the time to complete both questionnaires. The results are split in the three parts – time to finish questions on: learn, perceive and evolve programs. The average time effort of the end-users is shown in Table X.

Also, Table X show that the total average time to complete the DSL study was 54 minutes and to complete the GPL questionnaire was more then 1 hour and 17 minutes, so the end-users needed 42.3% more time to complete the questionnaire on GPL. It can be concluded that the bigger program length (see Table VIII) requires more time to understand, perceive and evolve GPL programs. The importance of a syntax [8], [9] should not be underestimated.

VI. CONCLUSION AND FUTURE WORK

As it does not exist any empirical study on advantages of DSLs over GPLs, the purpose of this paper was to compare them on program understanding. Questionnaires for hypothesis have been prepared and given to the end-users. Each end-user answered questionnaires in 86 pages and on average spent more then 2 hours solving 44 questions. The experiment sample included 38 end-users.

In the paper we present the empirical comparison on DSL and GPL programs understanding. The end-user success rate on questionnaires were on average 15% better in all three categories: learn, perceive and evolve. Also, end-users average time to complete the DSL questionnaire was 42.3% better then time to solve the GPL questionnaire. The standard metric eLOC was used to achieve fair comparison among equal DSL and GPL programs and shows that DSL programs were much shorter than GPL programs. All metrics show that program understanding is more efficient by DSL.

We consider that the results of this experiment are reliable despite that the experiment has been done only on one domain (DOT). One of the future tasks of this project, aiming at consolidating those achievements, is to do similar experiments in different domains. So, we will repeat this experimental study, with DSLs and application libraries implementations for the following two domains: domain feature descriptions (using FDL [4]) and construction of graphical user-interfaces (using XAML and C# Forms).

Another project direction is to include cognitive dimension framework (CDF) [7] – to identity the aspects among the CDF that enhanced in the context of DSL over GPLs. We need to study which dimensions are relevant or not for a DSL. So far, the CDF has been used in visual programming languages to assess their usability, while no such study exists for DSLs and GPLs.

Another objective of the work under discussion is to identify the precise needs in terms of information and visualization to comprehend DSL programs, in order to know if the existing approaches and techniques for the comprehension of GPL programs can be reused. Just as happens with program understanding tools, the tools for domain-specific program comprehension have to extract and display static or dynamic data about a program to help the analyst on understanding its structure and behavior.

REFERENCES

[1] V. Basili and F. Shull and F. Lanubile, *Building Knowledge through Families of Experiments*, IEEE Transactions on Software Engineering 25(4) 456–473, 1999.

[2] R. Brooks, *Using a behavioral theory of program comprehension in software engineering*, In Proceedings of the 3rd international conference on Software engineering, Piscataway, NJ, USA, IEEE Press 196–201, 1978.

[3] J. Carver, L. Jaccheri, S. Morasca and F. Shull, *A Checklist for Integrating Student Empirical Studies with Research and Teaching Goals*, To appear in Empirical Software Engineering, doi: 10.1007/s10664-009-9109-9.

[4] A. van Deursen, and P. Klint, *Domain-Specific Language Design Requires Feature Descriptions*, Journal of Computing and Information Technology 10(1) 1–17, 2002.

[5] Dot – Graph Description Language, *Available at:* http://en.wikipedia.org/wiki/DOT_language

[6] GraphViz – Graph Visualization Software, *Available at:* http://www.graphviz.org/

[7] T. Green and M. Petre, *Usability analysis of visual programming environments: a "cognitive dimensions" framework*, Journal of Visual Languages and Computing 7(2) 131–174, 1996.

[8] T. Kosar, P. E. Martínez López, P. A. Barrientos and M. Mernik, *A Preliminary Study on Various Implementation Approaches of Domain-Specific Language*, Information And Software Technology 50(5) 390–405, 2008.

[9] M. Mernik and J. Heering and A. Sloane, *When and How to Develop Domain-Specific Languages*, ACM Computing Surveys 37(4) 316–344, dec 2005.

[10] Categorized Lists of Computer Programming Languages, *Available at:* http://en.wikipedia.org/wiki/List_of_programming_languages

[11] Collection On Computer Programming Languages, *Available at:* http://www.people.ku.edu/ nkinners/LangList/Extras/langlist.htm

[12] L. Prechelt, *An Empirical Comparison of Seven Programming Languages*, IEEE Computer 33(10) 23–29, 2000.

[13] F. Shull, J. Carver, S. Vegas and N. Juristo, *The Role of Replications in Empirical Software Engineering*, Empirical Software Engineering 13(2) 211–218, 2008.

[14] M. A. Storey, *Theories, methods and tools in program comprehension: Past present and future*, In Proceedings of the 13th International Workshop on Program Comprehension (IPWC'05), pp. 181–191, 2005.

[15] M. J. Varanda Pereira, M. Mernik, D. da Cruz and P. R. Henriques *Program Comprehension for Domain-Specific Languages*, Journal on Computer Science and Information Systems, 5(2) 1–17, dec 2008.

[16] A. Walenstein, *Theory-based Analysis of Cognitive Support in Software Comprehension Tools*, In Proceedings of the 10th International Workshop on Program Comprehension (IWPC'02), pp. 75–84, 2002.

[17] Ž. Živanov, P. Rakić and M. Hajduković *Using Code Generation Approach in Developing Kiosk Applications*, Journal on Computer Science and Information Systems, 5(1) 41–59, jun 2008.