Comparing General-Purpose and Domain-Specific Languages: An Empirical Study

Tomaž Kosar[1], Nuno Oliveira[2], Marjan Mernik[1], Maria João Varanda Pereira[3], Matej Črepinšek[1], Daniela da Cruz[2], and Pedro Rangel Henriques[2]

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

Abstract. Many domain-specific languages, that try to bring feasible alternatives for existing solutions while simplifying programming work, have come up in recent years. Although, these little languages seem to be easy to use, there is an open issue whether they bring advantages in comparison to the application libraries, which are the most commonly used implementation approach. In this work, we present an experiment, which was carried out to compare such a domain-specific language with a comparable application library. The experiment was conducted with 36 programmers, who have answered a questionnaire on both implementation approaches. The questionnaire is more than 100 pages long. For a domain-specific language and the application library, the same problem domain has been used – construction of graphical user interfaces. In terms of a domain-specific language, XAML has been used and C# Forms for the application library. A cognitive dimension framework has been used for a comparison between XAML and C# Forms.

Keywords: domain-specific languages; general-purpose languages; program comprehension; empirical software engineering.

1. Introduction

The primary goal in developing a new programming language is to make programming more efficient. The perfect programming language should provide the right level of abstraction, meaning that it describes solutions naturally and hides unnecessary details. Also, it should be expressive enough in the problem domain and should provide guarantees on properties that are critical for the problem domain. It should also have precise semantics to

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

enable formal reasoning about a program. With general-purpose languages (GPLs), this is difficult to achieve, since GPLs tend to be general, resulting in poor support for domain-specific notation. On the other hand, domain-specific languages (DSLs) can be designed in many problem domains to exactly have properties mentioned above. A DSL is a language that is tailored to a specific application domain that offers appropriate notations and abstractions [1]. For a domain in question, DSLs are more expressive and are easier to use than GPLs, with gains in productivity and maintenance costs [2 - 4].

GPLs are perfectly established in the life-cycle of software development. Their characteristics are widely spread amongst software engineers. On the other hand, the integration of DSLs into the software development life-cycle is not so smooth [5]. However, many DSL studies during the last ten years [1, 3, 6 – 12], reveal the importance of these languages in software engineering. The concentration on a definition of notation that would only express concepts of a single application domain, brings the possibility to sharpen the edges of a language, which makes it more and more *efficient* in various directions, which are briefly elaborated below. One of these directions is the efficiency of being read and learned by the domain experts [13]. To use DSLs that allow focusing on the problem and not on the solution, can be profitable at earlier stages of the software life-cycle as well [14], such as requirements analysis and management [15]. Moreover, there is the possibility of integrating domain experts in the later stages of the software development life-cycle [2, 16]. Since the usage of GPLs requires good programming skills, the domain-experts, who are not proficient in that area, can do very little on this matter. However, with the use of DSLs, they can concentrate on the programming tasks and they can even do programming. Another benefit of DSLs is that software maintenance is simplified [2], since DSLs provide self-documentation that avoids the search for documentation resources, which may be unavailable in the first place. DSLs are also claimed to be a good approach for software reuse [17]. In this context, not only the pieces of software are reused, but also the knowledge embodied in the language. Another facet of efficiency can be observed in the tools that give support to a language. Their processors, for instance, can be improved to offer better results, as the domain is restricted and the knowledge is centralized [18, 19]. All together, these aspects diminish the costs of engineering and reengineering, and increase reliability and maintainability of the software constructed with DSLs [20].

Although, DSLs have proven their usefulness, GPLs together with application libraries (APIs) are still the most commonly used programmer's choice when preparing new solutions for their problems. One of the reasons that DSLs are not accepted among the practitioners is the lack of DSLs' promotion. Further, studies that would point out the benefits of DSL over GPL solution are rare. In this paper, we will use the cognitive dimension framework (CDF) [21, 22, 23] to compare DSL and GPL programs and to expose

properties that are enhanced in the context of DSLs. The goal of the project[1] is to measure how easy it is to understand programs written in DSLs compared to GPLs. In this manner, the experiment is conducted with the use of questionnaires to measure programmers' understanding of DSL and GPL programs on the same problem domain, a construction of graphical user interfaces (GUIs). More precisely, with these questionnaires, we attempt to confirm that DSL programs are easier to understand than GPL programs. This hypothesis is defended with an experiment in a controlled environment, using direct observations of the experiment evaluation model involving CDF.

The organization of this paper is as follows. Related work on the preparation of an experiment and CDF is discussed in Section 2. The experiment skeleton, the identification of its main goals, and experiment details are introduced in Section 3. The experiment results, with the cognitive dimension framework, are given in Section 4. Concluding remarks are summarized in Section 5.

2.   Related work

This work can be classified within the category of empirical software engineering. Empirical research in software engineering is an important discipline that shows practical results on how practitioners (developers, end-users) come to accept and use technologies, techniques, etc. In order to avoid questionable results and to have an option to repeat the research, giving the same results, experiments must be prepared with caution. One of the most well known frameworks for software experiments is described in [24]. This framework concentrates on building the knowledge concerning the context of an experiment and is based on organizing sets of related studies (family of studies). Such studies contribute to common hypotheses, which do not vary for individual experiments. In order to prepare this experiment we have followed guidelines from a framework [24]. We have also defined: context of the study, experiment hypothesis, comparison validity, and measurement framework.

Teaching environments give us an opportunity to conduct experiments in computer science programs as well. However, a lot of concerns are connected with the accuracy of results in such environments and several threats to the validity of experiments have to be identified as well as to interpret the results correctly. For those who are interested to read more about this topic, a checklist for integrating empirical studies in teaching activities can be found in the work [25].

As stated above, an important step in the experiment preparation is to set down the measurement framework – how the results of an experiment are evaluated and interpreted. In cognitive theory, guidelines on how to measure

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

a human's ability to program are defined. CDF [21] provides cognitively-relevant aspects which can be used to determine how easy it is to understand a program. In our study, CDF is used to compare user understanding of DSL and GPL programs. In the past the CDF has been used to assess the usability of visual programming languages [26, 27] and spreadsheets [28].

Recently, another application for cognitive dimensions can be found in [29], where a method for designing Framework-Specific Modeling Languages (FSMLs) is presented. From FSML specifications, a user can build applications based on object-oriented frameworks. In FSML software, artifacts (models, languages, etc.) are evaluated according to their goals with different quality methods. Particularly, the quality of notation is measured with cognitive dimensions – a heuristic measure that evaluates the notation and its environment.

Before this experiment, the authors of the paper were involved in another similar experiment [30]. That work is important for an interested reader, since the information on experiment skeletons is described in great detail. Difference between both experiments is in the hypothesis and exclusion/inclusion of CDF. Also, the problem domain in experiment [30] is different (graph description with DOT language [31]) than in this paper (construction of GUI with XAML).

This paper is also closely related to the field of Program Comprehension, which is a hard cognitive task, done by a software analyst. In the process of program comprehension, the use of tools to interconnect different views (operational, behavioral, etc.) to understand the results of applications, are indispensable. Traditional techniques on program comprehension from GPLs (visualizers, animators, etc.) have been studied and applied to DSLs in our previous work [32], where CDF was also briefly described and applied to DSLs.

3. Presentation of experiment

In this section the preparation, execution, and experiment evaluation model is given.

3.1. Objective of the experiment

In [3] the empirical results that compare ten diverse implementation approaches for DSLs, conducted on the same representative language, are provided. Among the implementation approaches, the comparison also included the XML-based approach. From this study, it can be concluded that XML-based approach has some disadvantages [3]. Although, XML usage and its tool support are spreading, this is one of the reasons that XAML [33], as a representative DSL, has been chosen for this study. XAML, the Extensible Application Markup Language, is a language for construction of graphical

user interfaces in Windows Presentation Foundation and Silverlight applications of .NET Framework 3.5. C# Forms [34] has been used for the comparison since it covers the same domain of graphical user interfaces.

3.2. Hypothesis of the experiment

In order to perform the comparison on XAML and C# Forms, two separate questionnaires have been prepared. The study that was carried out had a task to observe programmers' efficiency on understanding programs with both approaches, compare the results obtained through questionnaires and use them to investigate the following hypothesis:

$H1_{null}$

There is no significant difference in program understanding between domain-specific or general-purpose languages, when using XAML or C# Forms for comparison.

$H1_{alt}$

There is a significant difference in program understanding between domain-specific or general-purpose languages when using XAML instead of C# Forms.

This hypothesis is the object of investigation in the conducted experiment and is further examined in the Section 4, where the questionnaires results are presented.

3.3. Preparation of experiment

The results from an experiment are reliable if the repetition of the experiment can be proven [35]. Repetition is strongly connected to agreements set down before the start of the experiment [24]. Therefore, some rules and constraints were defined for the questionnaire implementers:

– the same group of questions for both experiments on a GPL and a DSL must be used,

– the questions for two applications on the same question were prepared (easier and harder application domain),

– the equal questions in DSL and GPL questionnaires must be defined by the same number of components in order 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.

As stated above, two questionnaires have been prepared for program understanding of DSL and GPL programs. Then, the structure of questionnaires has been defined to cover the following three topics of program understanding: learn, perceive, and evolve. In the first group, questions on learning notation and meaning of programs have been given to the programmers. In the second group, questions on program perceiving

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

have been defined, such as identification of correct meaning from a given program, language constructs, new construct meaning, and meaning of a program with given comments. In the third group, programmers 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 a valid program with the given result.
– Perceive
• Q4 Select a correct result for the given program.
• Q5 Identify language constructs.
• Q6 Select a program with the same result.
• Q7 Select a 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 in the program.

Learning and perceiving questions have been defined as a multiple-choice question, and questions under evolve have been defined as an essay question (programmers are challenged to modify existing code). Both, XAML and C# Forms questionnaires have been constructed with the use of the above questions.

To illustrate the style of the questions, used in the questionnaires, an example is presented in Figure 1. Because of the question size only the correct choice is given. Complete questionnaires can be found on a project group webpage[2]. The above questions (Q1-Q11) have been used as templates to define DSL and GPL questionnaires. The first version of DSL and GPL questionnaires has been given to a small group as a training set, in order to receive feedback. Further, results from a training set have been studied and used to refine the questions before applying them to the target groups of students. Most correctness issues were related to the program failures, question understandability and question complexity comparability between DSL and GPL questions.

3.4. Execution of the experiment

Besides well-structured questionnaires, other factors have also been controlled in the experiment. The list of experiment execution actions in the classroom (just before starting, as soon as it begins, and during the experiment) are provided below:
– a short tutorial, for end-users, has been given on the problem domain (graphical user interfaces),

---

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

– a tutorial on domain specific notation (XAML), together with an example of a program, has been given to end-users,

– a tutorial on application library (C# Forms) together with an example of the program has been given to end-users,

– a tutorial has been given to end-users in their native language, however the slides, programs and experiment questionnaires were in English, and

– the slides and the examples have been given to end-users and could be used during the experiment.

If necessary, individual help to better understand the questions was provided to the programmers.

3.5.　Processing the results

There were also issues that needed to be controlled, after the programmers completed the questionnaire. One of those is submission completeness. When students submitted their questionnaires, it was checked if the questionnaire contained answers to all questions. Most of the programmers answered the questions, however if some answers were missing, the programmers were advised to complete the questionnaire. Still, if some answers were found missing during the processing of the results, the complete programmer questionnaire was eliminated from the further experiment analysis. For previously mentioned reasons, one submission has been eliminated from the results.

3.6.　Threats to validity

In each experiment, there are several threats to the validity of results. Those threats need to be identified and handled before the start of the experiment. To restrict the impact of the experiment environment on the results, the following issues have been identified for our study.

Chosen domain Results of the experiment are strongly connected to programmers' experiences and knowledge of the chosen problem domain. In Table 1, programmers' familiarity with the construction of the GUI is presented, together with the experience on XAML and C# Forms library application. From Table 1, we can conclude that programmers are experienced in the construction GUIs domain. However, their experience in implementation technique differs – programmers were unfamiliar with XAML on one hand (median value 1), and had good knowledge in constructing GUIs with C# Forms (median value 4) on the other. Uneven knowledge on both notations could have made an influence on comparison results.

Programmers' experience In Table 2, results from the self evaluation test are presented, where students (second year of undergraduate computer

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

science) grade their general knowledge on programming, programming in C# language and prior experience with DSLs. Comparing knowledge on C# (median value 4) and prior experience with DSLs (median value 2) could have also made an influence on experiment results.

Table 1. Programmers' knowledge in construction of graphical user interfaces (N = 36)

	Average[3]	Median	St. dev.
Familiarity GUI domain	3.39	4	1.18
Knowledge of XAML	1.36	1	0.68
Knowledge of C# Forms application library	3.5	4	1.11

Table 2. Programmers experiences in programming (N = 36)

	Average	Median	St. dev.
Skills in programming	3.41	3.5	0.65
Skills of programming in C#	3.53	4	0.74
Prior experience with DSLs	2.28	2	0.70

Comparability of questionnaires The same type of questions in DSL and GPL questionnaires contain a similar number of graphical components (labels, text fields, buttons, etc), to obtain the same level of complexity.

Order of questionnaires An experiment on program understanding was carried out twice, at different times and on different students. The first group started the experiment with the questionnaire on DSL and proceeded with a GPL questionnaire. The second group started the questionnaire on GPL and finished with the DSL questionnaire. In such a way, the influence of starting the experiment on the same questionnaire with all subjects was avoided and with such, the order of questionnaires is not relevant for the outcome of the study.

4. Results

All together, programmers answered 22 questions on both questionnaires. Success rate for questions varied from 27.14% for Q6 to 79.73% for Q9 (Table 3). Differences in success rate in the same language (DSL/GPL) can be explained with different difficulty level (some questions were harder than

---

[3] A five-grade scale, starting from very bad (1) to very good (5) was used for self-evaluation questionnaires (in Tables 1 and 2). Note, that column "Average" shows the average value given by 36 programmers, "Median" stands for middle value in set of programmers grades and "St. dev." represents standard deviation on given grades.
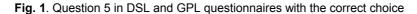
Question 5

QL031 XAML-DSPL-RegistrationForm: Select program for the following figure:

```
<Window x:Class="WpfRegistration.Registration3"
  Title="Registration form 3" Height="200" Width="300">
  <Grid ShowGridLines="False">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="12*"/> <ColumnDefinition Width="2*"/>
      <ColumnDefinition Width="10*"/> </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="10*"/> <RowDefinition Height="10*"/>
    <RowDefinition Height="10*"/> <RowDefinition Height="10*"/>
    <RowDefinition Height="10*"/> <RowDefinition Height="10*"/>
    <RowDefinition Height="10*"/> </Grid.RowDefinitions>
  <Label    Grid.Column="0" Grid.Row="1"> Username:</Label>
  <TextBox Grid.Column="2" Grid.Row="1" Background="LightGray"/>
  <Label    Grid.Column="0" Grid.Row="2"> Full name:</Label>
  <TextBox Grid.Column="2" Grid.Row="2"/>
  <Label    Grid.Column="0" Grid.Row="3"> Password:</Label>
  <TextBox Grid.Column="2" Grid.Row="3" Background="LightGray"/>
  <Label    Grid.Column="0" Grid.Row="4"> Retype password:</Label>
  <TextBox Grid.Column="2" Grid.Row="4" Background="LightGray"/>
  <Button  Grid.Column="2" Grid.Row="5"> Register</Button> </Grid> </Window>
```

Question 5

QL031 WFORM-GPL Select the correct program to produce the following figure:

```
private Label labelNumber, labelFilledNumber;
private Label labelInf, labelPic;
private TextBox textBoxInfo, textBox1;
private Button btnBrowse;

private void InitializeComponent() {
  labelNumber = new Label();
  labelFilledNumber = new Label();
  textBoxInfo = new TextBox();
  labelInf = new Label(); labelPic = new Label();
  btnBrowse = new Button(); textBox1 = new TextBox();
  labelNumber.ForeColor = System.Drawing.Color.Red;
  labelNumber.Location = new System.Drawing.Point(12, 9);
  labelNumber.Name = "labelNumber";
  labelNumber.Size = new System.Drawing.Size(119, 17);
  labelNumber.TabIndex = 5;
  labelNumber.Text = "Product Number: ";
  labelFilledNumber.Name = "labelFilledNumber";
  labelFilledNumber.Size = new System.Drawing.Size(72, 17);
  labelFilledNumber.Text = "90053918";
  labelFilledNumber.Location = new System.Drawing.Point(137, 9);
  textBoxInfo.Location = new System.Drawing.Point(109, 74);
  textBoxInfo.Multiline = true; textBoxInfo.Name = "textBoxInfo";
  textBoxInfo.Size = new System.Drawing.Size(246, 97);
  textBoxInfo.Text = "Price : 49,9 €\r\nAssembled size\r\nWidth: 40 cm\r\nDepth: 48 cm\r\nHeight: 56 cm";
  labelInf.Location = new System.Drawing.Point(12, 74);
  labelInf.Name = "labelInf";
  labelInf.Size = new System.Drawing.Size(82, 17);  labelInf.Text = "Information:";
  labelPic.Location = new System.Drawing.Point(12, 190);
  labelPic.Name = "labelPic";
  labelPic.Size = new System.Drawing.Size(56, 17);  labelPic.Text = "Picture:";
  btnBrowse.BackColor = System.Drawing.Color.Yellow;
  btnBrowse.Location = new System.Drawing.Point(276, 190);
  btnBrowse.Name = "btnBrowse";
  btnBrowse.Size = new System.Drawing.Size(79, 20);  btnBrowse.Text = "Browse";
  btnBrowse.UseVisualStyleBackColor = false;
  textBox1.Location = new System.Drawing.Point(109, 190);
  textBox1.Name = "textBox1";
  textBox1.Size = new System.Drawing.Size(161, 20);
  Controls.Add(textBox1); Controls.Add(btnBrowse); Controls.Add(labelPic); Controls.Add(labelInf);
  Controls.Add(textBoxInfo); Controls.Add(labelFilledNumber); Controls.Add(labelNumber);
}
```

Fig. 1. Question 5 in DSL and GPL questionnaires with the correct choice

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

others). On the other hand the biggest difference between GPL and DSL is 51.16% in the case of Q9. The smallest difference is found in Q2, where the difference is just 3.44%. In this case, the success rate was even slightly better for GPL than DSL. In our opinion, this is due to the difficultness of Q2 (success rate was less than 39%), where syntactically correct programs with no sense have to be identified. Since programmers have more experience in C# Forms than XAML (Table 1), they were more successful with GPL than DSL, in finding programs with no sense.

Table 3. Average programmer success rate (N = 35)

Question	DSL	GPL	Difference
	XAML	C# Forms	
Q1	72.97%	48.57%	24.4%
Q2	35.14%	38.57%	-3.44%
Q3	64.86%	35.71%	29.15%
Q4	77.03%	70.00%	7.03%
Q5	64.86%	48.57%	16.29%
Q6	39.19%	27.14%	12.05%
Q7	75.68%	62.86%	12.82%
Q8	62.16%	45.71%	16.45%
Q9	79.73%	28.57%	51.16%
Q10	68.92%	41.43%	27.49%
Q11	66.22%	30.00%	36.22%

Table 4. Average programmer success rate on learn, perceive and evolve (N=35)

	Question	DSL XAML			GPL C# Forms		
		Mean	Std. dev.	Std. err. mean	Mean	Std. dev.	Std. err. mean
Learn	Q1, Q2, and Q3	57.62%	21.90%	3.70%	40.95%	22.99%	3.89%
Perceive	Q4, Q5, Q6, Q7, and Q8	64.57%	19.45%	3.29%	50.86%	18.69%	3.16%
Evolve	Q9, Q10, and Q11	70.95%	20.35%	3.44%	33.33%	26.20%	4.43%
Total	All questions	64.34%	14.81%	2.50%	43.37%	15.99%	2.70%

However, drawing conclusions based on an average value of a single question can be extremely risky. Therefore, by grouping questions into learn, perceive and evolve categories, we can obtain more reliable results. In Table 4, the success rate on questions by the individual group is presented with a mean value, standard deviation, and standard error mean. Table 4 confirms our presumption that program understanding, in terms of learn, perceive and evolve, is much better for DSL programs than for GPL programs. Later observation is especially obvious from the results on evolve questions – the mean value of the success rate was 37.62% better for DSL than on GPL

questions (see mean values in Table 4). Similar results were also obtained on the other problem domain described in [30]. To support the results in Table 3 and Table 4, statistical tests have been performed to evaluate whether the comparison shows the statistical significant difference. Efficiency on both questionnaires (see last row from Table 4) has been compared for all programmers. The results from questionnaires were statistically tested with t-test, since the means of two independent groups were compared. The threshold for the independent t-test was set to α = 0.05 and the test results are shown in Table 5. The most important column in the table is the significance

Table 5. Independent t-test for program understanding (N = 35)

	t	Sig. (two-tailed)	Mean	Std. dev.	Std. err. mean	95% Confidence Interval of the Difference	
						Lower	Upper
XAML vs. C# Forms	5.474	0.000	20.971	22.664	3.831	13.186	28.757

column. Observing this data confirms that the difference in mean value between XAML and C# Forms program understanding was significant, since a significant level was not reached. With observations from Table 5, we could reject null hypothesis $H1_{null}$ since subjects did better on XAML program understanding and accept the alternative hypothesis: ($H1_{alt}$): there is a significant difference in program understanding between domain-specific or general-purpose languages when using XAML instead of C# Forms.

While this experiment indeed shows superiority of DSLs on an end-user ability to learn, perceive and evolve programs in this particular domain, it does not provide possible explanations why DSLs programs are easier to understand. The "psychology of programming" [36, 37] is a research field which tries to identify, understand and explain those cognitive processes which take place during reasoning (e.g., programming, program understanding). In this context, CDF [21] provides useful dimensions, which help us to better explain why DSL programs are easier to understand than GPL programs. The CDF has been used before to assess the usability of visual programming languages [26], while no such study exists for DSLs. These cognitive dimensions are:
  – Closeness of mapping – languages should be task-specific;
  – Viscosity – revisions should be painless;
  – Hidden dependencies – the consequences of changes should be clear;
  – Hard mental operations – no enigmatic is allowed;
  – Imposed guess-ahead – no premature commitment;
  – Secondary notation – allow to encompass additional information;
  – Visibility – search trails should be short;
  – Consistency – user expectations should not be broken;

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

    – Diffuseness – language should not be too verbose;
    – Error-proneness – notation should catch mistakes avoiding errors;
    – Progressive evaluation – get immediate feedback;
    – Role expressiveness – see the relations among components clearly;
    – Abstraction gradient – languages should allow different abstraction
levels.

The next step was to connect cognitive dimensions with our questions. We
identified which dimensions are relevant for a particular question (Table 6). As
it can be seen D_i (dimension i of CDF) can be related to several questions
used in our questionnaires.

Table 6. Questions connection to cognitive dimensions

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Closeness of mapping	1	1	1	1	1	1	1	1	1	1	1
Viscosity	0	0	0	0	0	0	0	0	1	1	1
Hidden dependencies	0	0	1	1	1	1	0	1	0	1	0
Hard mental operations	0	1	1	1	1	1	1	1	0	0	0
Imposed guess-ahead	0	0	0	0	0	0	0	0	1	0	1
Secondary notation	0	0	0	0	0	0	0	1	0	0	0
Visibility	0	0	1	1	1	1	1	1	0	0	0
Consistency	0	0	0	0	0	1	1	0	0	0	0
Diffuseness	1	1	1	1	1	1	1	1	1	1	1
Error-proneness	1	1	1	1	1	1	1	1	1	1	1
Progressive evaluation	0	0	0	0	0	0	0	0	0	0	0
Role expressiveness	0	1	1	1	1	1	1	1	1	1	1
Abstraction gradient	0	0	1	1	1	1	1	1	0	0	0

Questions have been designed in such a way that they directly reflect
cognitive dimensions as much as possible. However, not all cognitive
dimensions play an important role in all questions. Hence, to evaluate a single
cognitive dimension (D_i) we proposed the following formula:

$$D_i = \sum_{j=1}^{11} Q_{ij} * \frac{S_j}{C_j}$$

where Q_{ij} stands for the value from Table 6, which means whether dimension
D_i is connected to the question Q_j. Variable S_j represents an average
programmer's success rate on question Q_j (Table 3). For example, if 4
programmers out of 5 answered question Q_1 correctly, the value of S_1 would
be 0.8. Finally, C_j represents the number of cognitive dimensions relevant for
Q_j (for example, $C_1 = 3$). This formula is used for XAML as well as for C#
Forms. Intuitively, it means that cognitive dimensions contribute to the
success of a particular question. Here, we assume that contribution of
involved cognitive dimensions was equally distributed (one cognitive
dimension is not more important than the other, if it is involved). Moreover, we
assume that the higher values always mean a positive influence of particular
cognitive dimension. For example, higher values for 'closeness of mapping'

mean that the semantic gap between the problem and the solution space is small, or higher values for 'hidden dependencies' mean that short and long-range interactions among program components are immediately visible.

Table 7 roughly shows how a particular cognitive dimension contributes to the questionnaires' success for XAML, as well as for C# Forms. From Table 7 it can be seen that in our experiment, the most influential for DSL/GPL program understanding were: closeness of mappings, diffuseness, error-proneness, role expressiveness, and hard mental operations. More than particular values, the difference among cognitive dimensions for XAML and C# Forms is far more important. The biggest difference among cognitive dimensions was in closeness of mappings, diffuseness, error-proneness, role expressiveness, and viscosity.

Table 7. Influence of cognitive dimension to XAML and C# Forms

	DSL	GPL	Difference
	XAML	C# Forms	
Closeness of mapping	1.127	0.749	0.377
Viscosity	0.442	0.237	0.206
Hidden dependencies	0.486	0.343	0.143
Hard mental operations	0.525	0.421	0.105
Imposed guess-ahead	0.243	0.098	0.146
Secondary notation	0.069	0.051	0.018
Visibility	0.455	0.344	0.111
Consistency	0.128	0.100	0.028
Diffuseness	1.127	0.749	0.377
Error-proneness	1.127	0.749	0.377
Progressive evaluation	N/A	N/A	N/A
Role expressiveness	0.884	0.587	0.296
Abstraction gradient	0.455	0.344	0.111

Closeness of mapping refers to the width of the semantic gap between the problem and the solution spaces. Diffuseness refers to the number of symbols needed to express the meaning. By definition, DSLs use existing domain notation, which should be at an appropriate level of verbosity, so it is expected that they exhibit low diffuseness. On the other hand, it was shown in [38] that plenty of low-level primitives, which are often purely syntactical, are one of the biggest cognitive barriers for end-user programmers. Error proneness refers to the capability of a language to induce careless mistakes. GPLs, due to their extension and intrinsic complexity, are usually error-prone, while DSLs, due to the narrow domain they are designed for, are usually less error prone. Role expressiveness refers to the ability to see how each component of a program relates to the whole. The high role expressiveness can be more easily achieved in DSLs due to domain specifics and shorter programs. It is shown in our experiment that differences in closeness of mapping, diffuseness, error proneness, and role expressiveness among

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

XAML and C# Forms are the biggest and the source of main contribution for easier understanding of XAML programs than programs written in C# Forms.

Viscosity refers to the amount of effort that is needed to perform small changes. Since DSLs are usually at a high abstraction level and have natural notation, small changes should be easier to perform. It is shown in our experiment that the difference in viscosity between XAML and C# Forms was among the largest. Viscosity was involved only in questions Q9-Q11, which were much better solved with the use of XAML than using C# Forms. We can conclude that viscosity had an important influence on this success.

5.   Conclusion and future work

The purpose of this paper is to promote formal studies on the advantages of DSLs over GPLs. In this paper we have tried to explain the difference between DSL/GPL program understanding, using the cognitive dimension framework. Questionnaires on understanding programs have been prepared and given to the programmers. Each programmer answered a 100 page long questionnaires and on an average spent more than 3 hours solving 44 questions.

Results show that programmers' success rate was around 15% better for DSL in all three groups of questions: learn, perceive and evolve, despite the fact that programmers were significantly less experienced in XAML than C# Forms. Further, the experiment measurement framework included cognitive dimensions to identify the aspects among these dimensions that are enhanced in the context of DSL. It can be learned from the study that DSLs are superior to GPLs in all cognitive dimensions. The cognitive dimensions, with the biggest influence in the experiment, are closeness of mappings, diffuseness, error-proneness, role expressiveness, and viscosity.

We consider that the results of this experiment are reliable despite the fact that the experiment has been done only on a single domain. One of the future tasks of this project is to conduct similar experiments in different domains.

References

1.   Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Computing Surveys. 37(4) (December 2005) 316–344
2.   Deursen, A.v., Klint, P.: Little languages: Little maintenance? Journal of Software Maintenance 10 (1998) 75–92
3.   Kosar, T., Martínez López, P.E., Barrientos, P.A., Mernik, M.: A preliminary study on various implementation approaches of domain-specific language. Information and Software Technology 50(5) (2008) 390–405
4.   Živanov, v., Rakić, P., Hajduković, M.: Using code generation approach in developing kiosk applications. Journal on Computer Science and Information Systems 5(1) (2008) 41–59

5. Sprinkle, J., Mernik, M., Tolvanen, J-P., Spinellis, D.: What Kinds of Nails Need a Domain-Specific Hammer? IEEE Software, 26(4) (2009) 15–18
6. Wile, D.S.: Supporting the DSL spectrum. Journal of Computing and Information Technology 9(4) (2001) 263–287
7. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices 35 (2000) 26–36
8. Hudak, P.: Building domain-specific embedded languages. ACM Computing Surveys 28(4) (June 1996) 196–202
9. Hudak, P.: Modular domain specific languages and tools. In: ICSR '98: Proceedings of the 5th International Conference on Software Reuse, Washington, DC, USA, IEEE Computer Society (1998)
10. Kieburtz, R.B., Mckinney, L., Bell, J.M., Hook, J., Kotov, A., Lewis, J., Oliva, D.P., Sheard, T., Smith, I., Walton, L.: A software engineering experiment in software component generation. In: ICSE '96: Proceedings of the 18th international conference on Software engineering, Washington, DC, USA, IEEE Computer Society (1996) 542–552
11. Sirer, E.G., Bershad, B.N.: Using production grammars in software testing. In: Proceedings of the 2nd conference on Domain-specific languages, New York, NY, USA, ACM (1999) 1–13
12. Kolovos, D.S., Paige, R.F., Kelly, T., Polack, F.A.C.: Requirements for domain-specific languages. In: Proc. 1st ECOOP Workshop on Domain-Specific Program Development (DSPD 2006), Nantes, France (July 2006)
13. Consel, C., Latry, F., Réveillère, L., Cointe, P.: A generative programming approach to developing DSL compilers. In Gluck, R., Lowry, M., eds.: Fourth International Conference on Generative Programming and Component Engineering (GPCE). Volume 3676 of Lecture Notes in Computer Science, Tallinn, Estonia, Springer-Verlag (September 2005) 29–46
14. Jackson, M.: Problem frames and software engineering. Information and Software Technology 47(14) (November 2005) 903–912
15. Aurum, A., Wohlin, C.: Engineering and Managing Software Requirements. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
16. Gray, J., Fisher, K., Consel, C., Karsai, G., Mernik, M., Tolvanen, J.P.: DSLs: the good, the bad, and the ugly. In: OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, New York, NY, USA, ACM (2008) 791–794
17. Krueger, C.W.: Software reuse. ACM Computing Surveys 24(2) (June 1992) 131–183
18. Javed, F., Mernik, M., Bryant, B., Sprague, A.: An unsupervised incremental learning algorithm for domain-specific language development. Applied Artificial Intelligence 22(7) (2008) 707-729
19. Wu, H., Gray, J., Mernik, M.: Grammar-driven generation of domain-specific language debuggers. Software Practice and Experience 38(10) (2008) 1073-1103
20. Herndon, R.M., Berzins, V.A.: The realizable benefits of a language prototyping language. IEEE Transactions on Software Engineering 14(6) (1988) 803–809
21. Green, T., Petre, M.: Usability analysis of visual programming environments: a "cognitive dimensions" framework. Journal of Visual Languages and Computing 7(2) (1996) 131–174
22. Blackwell, A., Britton, C., Cox, A., Green, T.R.G., Gurr, C., Kadoda, G., Kutar, M., Loomes, M., Nehaniv, C., Petre, M., Roast, C., Roe, C., Wong, A., Young, R.: Cognitive dimensions of notations: Design tools for cognitive technology. In: Cognitive Technology: Instruments of Mind. Springer-Verlag (2001) 325–341

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

23. Green, T.R.G., Blandford, A.E., Church, L., Roast, C.R., Clarke, S.: Cognitive dimensions: achievements, new directions, and open questions. Journal of Visual Languages & Computing 17(4) (August 2006) 328–365
24. Basili, V., Shull, F., Lanubile, F.: Building knowledge through families of experiments. IEEE Transactions on Software Engineering 25(4) (1999) 456–473
25. Carver, J., Jaccheri, L., Morasca. S., Shull. F.: A checklist for integrating student empirical studies with research and teaching goals. To appear in Empirical Software Engineering (2009)
26. Blackwell, A.: Ten years of cognitive dimensions in visual languages and computing: Guest editor's introduction to special issue. Journal of Visual Languages and Computing 17(4) (2006) 285–287
27. Yang, S., Burnett, M., DeKoven, E., Zloof, M.: Representation design benchmarks: a design-time aid for VPL navigable static representations. Journal of Visual Languages and Computing 8(5/6) (1997) 563–599
28. Peyton Jones, S., Blackwell, A., Burnett, M.: A user-centred approach to functions in excel. In: Proceedings of the eighth ACM SIGPLAN international conference on Functional programming. (2003) 165–176
29. Antkiewicz, M., Czarnecki, K., Stephan, M.: Engineering of framework-specific modeling languages. IEEE Transactions on Software Engineering 35 (6) (2009) 795–824
30. Kosar, T., Mernik, M., Črepinšek, M., Henriques, P.R., Cruz, D.d., Varanda Pereira, M.J., Oliveira, N.: Influence of domain-specific notation to program understanding. In: Proceedings of 2nd IMCSIT Workshop on Advances in Programming Languages (WAPL'09), Mrągowo, Poland (October 2009) 675 – 682
31. Dot: Graph description language, available at: http://en.wikipedia.org/wiki/DOT language
32. Varanda Pereira, M.J., Mernik, M., Cruz, D.d., Henriques, P.R.: Program comprehension for domain-specific languages. Journal on Computer Science and Information Systems 5(2) (2008) 1–17
33. XAML: Extensible application markup language, available at: http://en.wikipedia.org/wiki/Extensible Application Markup Language
34. C# Forms, available at: http://en.wikipedia.org/wiki/Windows Forms
35. Shull, F., Carver, J., Vegas, S., Juristo, N.: The role of replications in empirical software engineering. Empirical Software Engineering 13(2) (2008) 211–218
36. Weinberg, G.M.: The Psychology of Computer Programming. Van Nostrand Reinhold (1971)
37. A. Blackwell. Psychological issues in end-user programming. In H. Lieberman, F. Paterno, and V. Wulf, editors, End User Development Springer (2006) 9–30
38. Lewis, C., Olson, G.: Can principles of cognition lower the barriers to programming? In: 2nd workshop on Empirical Studies of Programmers. (1987) 248 – 263

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 visual languages, empirical software engineering, software security, generative programming, compiler construction, object oriented programming, object-oriented design, refactoring, and unit testing. He is currently a teaching assistant at the

University of Maribor, Faculty of Electrical Engineering and Computer Science.

Nuno Oliveira received, from University of Minho, a B.Sc. in Computer Science (2007) and a M.Sc. in Informatics (2009). He is a member of the Language Processing group at CCTC (Computer Science and Technology Center) , University of Minho. He participated in several projects with focus on Visual Languages and Program Comprehension; VisualLISA and Alma2 the main outcome of his master thesis entitled "Program Comprehension Tools for Domain-Specific Languages", are the most relevant works. The latter came under "Program Comprehension for Domain-Specific Languages", a bilateral project between Portugal and Slovenia, funded by FCT. Currently, he is starting his PhD studies on Patterns for Architectures Coordination Analysis and Self-Adaptive Architectures, under MathIS, a research project also funded by FCT. Meanwhile he is an assistant-lecturer (practical classes) in a course on Imperative Programming, at University of Minho.

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 an adjunct professor at the University of Alabama at Birmingham, Department of Computer and Information Sciences. His research interests include programming languages, compilers, grammar-based systems, grammatical inference, and evolutionary computations. He is a member of the IEEE, ACM and EAPLS.

Maria João Varanda Pereira received the M.Sc. and Ph.D. degrees in computer science from the University of Minho in 1996 and 2003 respectively. She is a member of the Language Processing group in the Computer Science and Technology Center , at the University of Minho. She is currently an adjunct professor at the Technology and Management School of the Polytechnic Institute of Bragança, on the Informatics and Communications Department and vice-president of the same school. She usually teaches courses under the broader area of programming: programming languages, algorithms and language processing. But also some courses about project management. As a researcher of gEPL, she is working with the development of compilers based on attribute grammars, automatic generation tools, visual languages and program understanding. She was also responsible for PCVIA project (Program Comprehension by Visual Inspection and Animation), a FCT funded national research project; She was involved in several bilateral cooperation projects with University of Maribor (Slovenia) since 2000. The last one was about the subject ``Program Comprehension for Domain Specific Languages".

Matej Črepinšek received the Ph.D. degree in computer science at the University of Maribor, Slovenia in 2007. His research interests include grammatical inference, evolutionary computations, object-oriented

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

programming, compilers and grammar-based systems. He is currently a teaching assistant at the University of Maribor, Faculty of Electrical Engineering and Computer Science.

Daniela da Cruz received a degree in "Mathematics and Computer Science", at University of Minho (UM), and now she is a Ph.D. student of "Computer Science" also at University of Minho, under the MAPi doctoral program. She joined the research and teaching team of "gEPL, the Language Processing group" in 2005. She is teaching assistant in different courses in the area of Compilers and Formal Development of Language Processors; and Programming Languages and Paradigms (Procedural, Logic, and OO). As a researcher of gEPL, Daniela is working with the development of compilers based on attribute grammars and automatic generation tools. She developed a completed compiler and a virtual machine for the LISS language (Language of Integers, Sequences and Sets - an imperative and powerful programming language conceived at UM). She was also involved in the PCVIA (Program Comprehension by Visual Inspection and Animation), a FCT funded national research project; in that context, Daniela worked in the implementation of "Alma", a program visualizer and animator tool for program understanding. Now she is working in the intersection of formal verification (design by contract) and code analysis techniques, mainly slicing.

Pedro Rangel Henriques got a degree in "Electrotechnical/Electronics Engineering", at FEUP (Porto University), and finished a Ph.D. thesis in "Formal Languages and Attribute Grammars" at University of Minho. In 1981 he joined the Computer Science Department of University of Minho, where he is a teacher/researcher. Since 1995 he is the coordinator of the "Language Processing group" at CCTC (Computer Science and Technologies Center).
He teaches many different courses under the broader area of programming: Programming Languages and Paradigms; Compilers, Grammar Engineering and Software Analysis and Transformation; etc. Pedro Rangel Henriques has supervised Ph.D. (11), and M.Sc. (13) thesis, and more than 50 graduating trainingships/projects, in the areas of: language processing (textual and visual), and structured document processing; code analysis, program visulaization/animation and program comprehension; knowledge discovery from databases, data-mining, and data-cleaning. He is co-author of the "XML & XSL: da teoria a prática" book, publish by FCA in 2002; and has published 3 chapters in books, and 20 journal papers.