Comparison of XAML and C# Forms using Cognitive Dimension Framework

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

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

Abstract. Many domain-specific languages arise in the past years, trying to bring feasible alternatives for existing solutions with purpose to simplify programmers work. Although these little languages seem to be easier to use, there is an open issue whether they bring advantages comparing to most commonly used implementation approach, application libraries. In this work we present an experiment, carried out to compare such domain-specific language with comparable application library. The experiment was conducted with 36 programmers, which were answering questions on more than 100 long pages on both implementation approaches. For domain-specific language and application library the same problem domain has been used – construction of graphical user interfaces. In terms of domain-specific language, XAML has been used and C# Forms for application library. For comparison of XAML and C# Forms cognitive dimension framework has been used.

1 Introduction

The primary goal of developing 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 critical for the problem domain. It should also have precise semantics to enable formal reasoning about a program. With general-purpose languages (GPLs) this is hard to achieve, since they tend to be general with consequence on poor support for domain-specific notation. On the other hand, domain-specific languages (DSLs) can be designed in many problem domains to have exactly above properties. DSL is a language tailored to specific application domain that offers appropriate notations and abstractions

[12]. DSLs are more expressive and are easier to use than GPLs for the domain in question, with gains in productivity and maintenance costs [6], [10], [18].

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 cognitive dimension framework (CDF) [8] to compare DSL and GPLs programs and to expose properties that are enhanced in the context of DSLs. The goal of the project[4] is to measure how easy is to understand programs written in DSLs than in GPLs. In this manner, experiment is conducted using questionnaires to measure programmers understanding of DSL and GPL programs on same problem domain, a construction of graphical user interfaces (GUIs). More precisely, with these questionnaires we try to confirm that DSL programs are easier to understand than GPL programs. This hypothesis is here defended with experiment under controlled environment, using direct observations of experiment evaluation model involving CDF.

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

2 Related work

The DSL is usually developed in the following phases: decision, analysis, design, implementation, and deployment as identified in work [12]. For the implementation part, following techniques have been identified for DSLs: preprocessing, embedding, compiler/interpreter, compiler generator, extensible compiler/interpreter, and commercial off-the-shelf. In order to implement a DSL, a programmer has to choose among these implementation approaches and, of course, the most suitable one should be chosen according to project influences. As defined above, one of the implementation techniques is also commercial off-the-shelf approach. Here, existing tools and/or notations can be used to a specific domain, e.g., XML-based DSLs. This approach provides a feasible alternative when solving particular domain problems and for instance, XML here brings promising solutions in processing and querying documents for data exchange, etc. In general, XML tends to be cumbersome for humans to read and write and has some other disadvantages comparing to other DSL implementation approaches [10], however it seems that the approach is very well accepted by the leading technologies in the software industry and it is not expected that this fact will change in the near future.

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

This work can be classified to empirical software engineering category. Empirical research in software engineering is an important discipline, showing practical results on how practitioners (developers, end-users) come to accept and use technologies, techniques, etc. In order to avoid questionable results and to enable repetition of research giving the same results, experiment has to be prepared with caution. One of the most well known framework for software experiments is described in [2]. This framework concentrates on building knowledge about the context of an experiment and is based on organizing sets of related studies (family of studies). Such studies contribute to a common hypotheses which does not vary for individual experiments. Therefore, we followed guidelines from framework [2] in order to prepare this experiment and we defined: context of the study, experiment hypothesis, comparison validity, and measurement framework.

Teaching environments give us an opportunity to conduct experiments also in computer science programs. However, a lot of concerns are connected with the accuracy of results in such environments and several threats to validity of experiment has to be identified and to interpret the results correctly. Interested reader, can find more about this topic in the work [4], where a checklist for integrating empirical studies in teaching activities can be found.

As stated above, important step in experiment preparation is to set down the measurement framework – how results of experiment are evaluated and interpreted. In cognitive theory, guidelines how to measure human's ability to program, are defined. CDF [8] 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. While before CDF has been used to assess the usability of visual programming languages [3], [17] and spreadsheets [13].

Another application of cognitive dimensions can be found recently in [1], where method for designing Framework-Specific Modeling Languages (FSMLs) is presented. From FMLS specifications user can build applications based on object-oriented frameworks. In FSML software artifacts (models, languages, etc.) are evaluated according to its goals with different quality methods. Particularly, quality of notation is measured with cognitive dimensions – a heuristic measure that evaluates the notation and its environment.

Before this experiment, authors of the paper were involved in another similar experiment [9]. That work is important for interested reader, since information on experiment skeleton is described in details. Difference among both experiments is in hypothesis and exclusion/inclusion of CDF. Also, the problem domain in experiment [9] is different (graph description with DOT language [7]) 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 software analyst. In the process of program comprehension, the use of tools to interconnect different views (operational, behavioral, etc.) to understand results of application, are indispensable. Traditional techniques on program comprehension from GPLs (visualizers, animators, etc.) have been studied and applied to DSLs in our previous work [15], 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 Subject of comparison

In the work [10] the empirical results comparing ten diverse implementation approaches for DSLs, conducted on the same representative language, are provided. Among implementation approaches, comparison included also XML-based approach. From this study can be concluded, that XML-based approach, has some disadvantages [10]. Although, XML usage and its tool support are spreading. This is one of the reasons that XAML [16], as a representative DSL, has been chosen for this study. XAML, the Extensible Application Markup Language, is a language for representation of graphical user interfaces in Windows Presentation Foundation and Silverlight applications of .NET Framework 3.5. C# Forms [5] has been used for comparison since it covers the same domain of graphical user interfaces.

3.2 Preparation of experiment environment

The results from an experiment are reliable if the repetition of experiment can be proven [14]. Repetition is strongly connected to agreements set down before starting the experiment [2]. Consistency of results in our experiment were obtained by creating rules and constraints for programmers: using well-structured questionnaires, domain tutorials and extra explanations in their native language, before starting answering on questions. Tutorial to programmers included presentation of problem domain (GUI), domain specific notation (XAML) and application library (C# Forms) together with examples of programs. Consistency of results were obtained also with rules for questionnaire implementors, which had to define the same group of questions for both experiment on GPL and DSL. Implementors were also advised to prepare questions for two applications (easier and harder application domain). More on preparation of this experiment can be found in [9].

3.3 Threats to validity of experiment

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

Chosen domain Results of the experiment are strongly connected to programmers' experiences and knowledge of chosen problem domain. In Table 1, programmers familiarity with construction of GUI is presented, together with experience on XAML and C# Forms library application. From Table 1, we can conclude that programmers are experienced in domain of construction GUIs. 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). Uneven knowledge on both notations could have influence on comparison results.

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

	Average[1]	**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**

N = number of received questionnaires

Programmers experience In Table 2, we present results from self evaluation test, where students (second year of undergraduate computer science) grade their general knowledge about 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 also have influence on experiment results.

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

Experiment questionnaires As stated before, two questionnaires have been prepared for program understanding of DSL and GPL programs. Then, the structure of questionnaires has been defined to cover 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 have been defined, such as identification of: correct meaning from given program, language constructs, new construct meaning, and meaning of program with given comments. In the third

[1] A five-graded scale, going 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.

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**

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 valid program with given result.
- Perceive
  - Q4 Select correct result for the given program.
  - Q5 Identify language constructs.
  - Q6 Select program with same result.
  - Q7 Select 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 has been defined as multiple choice question, and questions on evolve has been defined as essay question (programmers are challenged to modify existing code). Both, XAML and C# Forms questionnaires have been constructed using the above questions.

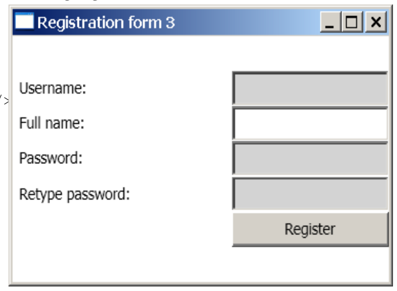
To illustrate the style of the questions used in the questionnaires, an example is presented in Figure 1. Because of question size only the correct choice is given. Complete questionnaires can be found at project group webpage[5].

4 Results

All together, programmers answered 22 questions on both questionnaires. Success rate for questions vary 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 others). On the other hand the biggest difference between GPL and DSL is 51.16% in case of Q9. The smallest difference we can find in Q2, where difference is just 3.44%. In this

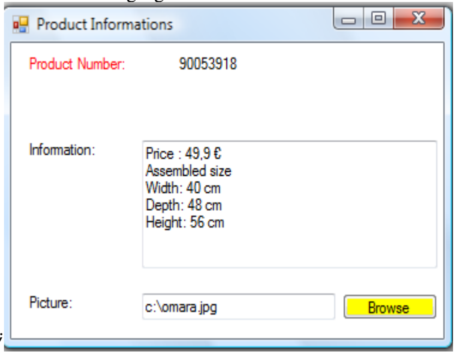
[5] http://epl.di.uminho.pt/∼gepl/DSL/

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:



90053918

Price : 49,9 €
Assembled size
Width: 40 cm
Depth: 48 cm
Height: 56 cm

c:\omara.jpg

```
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 correct choice

Table 3. Average programmer success rate (N = 36)

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 on learn, perceive and evolve (N = 36)

	Question	**DSL**	**GPL**	**Difference**
		XAML	**C# Forms**	
Learn	**Q1, Q2, and Q3**	**57.66%**	**40.95%**	**16.71%**
Perceive	**Q4, Q5, Q6, Q7, and Q8**	**63.78%**	**50.86%**	**12.93%**
Evolve	**Q9, Q10, and Q11**	**71.62%**	**33.33%**	**38.29%**

case success rate was even slightly better for GPL than DSL. To our opinion this is due to 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 for GPL than DSL in finding programs with no sense.

However, drawing conclusions based on average value of single question can be extremely risky. Therefore, by grouping questions on learn, perceive and evolve we can obtain more reliable results. In Table 4 average success rate on questions by individual group are presented. Table 4 confirms our presumption that program understanding in terms of learn, perceive and evolve is much better for DSL programs than on GPL programs. Later observation is specially obvious from results on evolve questions – success rate on this question was 38.29% better for DSL than on GPL questions. Similar results were also obtained on other problem domain described in [9].

One of possible explanation, why programs written in DSLs are easier to understand than programs written in GPLs, can be offered by CDF [8]. The CDF has been used before to assess the usability of visual programming languages [3], 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;
– 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 particular question (Table 5). As it can be seen Di (dimension i of CDF) can be related with several questions used in our questionnaires.

Table 5. 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**

To evaluate single cognitive dimension (Di), the following formula has been used:

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

where Q_{ij} stands for value from Table 5 meaning whether dimension Di is connected with question Q_j. Variable S_j represents average programmers success rate on question Q_j (Table 3). For example, if 4 programmers out of 5 answered correctly on question Q_1, 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 particular question. Here, we assume that contribution of involved cognitive dimensions were equally distributed (one cognitive dimension is not more important than other if it is involved). Moreover, we assume that higher values always mean positive influence of particular cognitive dimension. For example, higher values for 'closeness of mapping' mean that the semantic gap between the problem and solution space is small, or higher values for 'hidden dependencies' mean that short and long-range interactions among program components are immediately visible.

Table 6 roughly shows how much particular cognitive dimension contribute to the questionnaires' success for XAML, as well as for C# Forms. From Table 6 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 important is difference among cognitive dimensions for XAML and C# Forms. The biggest difference among cognitive dimensions were at closeness of mappings, diffuseness, error-proneness, role expressiveness, and viscosity.

Table 6. 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 how wide the semantic gap is between the problem and 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 the study [11] that plenty of low-level primitives, which are often purely syntactical, is 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 of closeness of mapping, diffuseness, error proneness, and role expressiveness among 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 how much effort is needed to perform small changes. Since DSLs are usually at 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 among XAML and C# Forms was among the biggest. Viscosity was involved only in questions Q9-Q11, which were much better solved using XAML and C# Forms. We can conclude that viscosity had an important influence to this success.

5 Conclusion and future work

The purpose of this paper is to promote formal studies on advantages of DSLs over GPLs. In this paper we tried to explain the difference among DSL/GPL program understanding using cognitive dimension framework. Questionnaires on understanding programs have been prepared and given to the programmers. Each programmer answered questionnaires written in more than 100 pages and on average spent more then 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 that programmers were significantly less experienced in XAML than C# Forms. Further, experiment measurement framework included cognitive dimensions to identify the aspects among these dimensions that are enhanced in the context of DSL. From the study can be learned that DSLs gain comparing 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 that experiment has been done only on single domain. One of the future tasks of this project, is to commit similar experiments in different domains.

References

1. Antkiewicz, M., Czarnecki, K., Stephan, M. *Engineering of Framework-Specific Modeling Languages*, To appear in IEEE Transactions on Software Engineering, 2009, http://doi.ieeecomputersociety.org/10.1109/TSE.2009.30.

2. Basili, V., Shull, F., Lanubile, F. *Building Knowledge through Families of Experiments*, IEEE Transactions on Software Engineering 25(4) 456–473, 1999.
3. Blackwell, A.F. *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), 285–287, 2006.
4. 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, doi: 10.1007/s10664-009-9109-9.
5. C# Windows Forms, *Available at:* http://en.wikipedia.org/wiki/Windows_Forms
6. Deursen, A. van, Klint, P. *Little languages: Little maintenance?*, Journal of Software Maintenance (10), 75–92, 1998.
7. Dot – Graph Description Language, *Available at:* http://en.wikipedia.org/wiki/DOT_language
8. Green, T., Petre, M. *Usability analysis of visual programming environments: a "cognitive dimensions" framework*, Journal of Visual Languages and Computing 7(2) 131–174, 1996.
9. Kosar, T., Mernik, M., Črepinsek, M., Henriques, P. R., Cruz, D. da, Varanda Pereira, M. J., Oliveira, N. *Influence of domain-specific notation to program understanding*, Submitted to 2nd Workshop on Advances in Programming Languages (WAPL'09), 2009.
10. 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) 390–405, 2008.
11. Lewis, C., Olson, G. *Can principles of cognition lower the barriers to programming?*, 2nd workshop on Empirical Studies of Programmers, 1987.
12. Mernik, M., Heering, J., Sloane, A. *When and How to Develop Domain-Specific Languages*, ACM Computing Surveys 37(4) 316–344, 2005.
13. Peyton Jones, S., Blackwell, A., Burnett, M. *A User-Centred Approach to Functions in Excel*, Proceedings of the eighth ACM SIGPLAN international conference on Functional programming, 165–176, 2003.
14. Shull, F., Carver, J., Vegas, S., Juristo, N., *The Role of Replications in Empirical Software Engineering*, Empirical Software Engineering 13(2) 211–218, 2008.
15. Varanda Pereira, M. J., Mernik, M., Cruz, D. da, Henriques, P. R., *Program Comprehension for Domain-Specific Languages*, Journal on Computer Science and Information Systems, 5(2) 1–17, 2008.
16. XAML – Extensible Application Markup Language, *Available at:* http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language
17. 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) 563–599, 1997.
18. Živanov, Ž., Rakić, P., Hajduković, M. *Using Code Generation Approach in Developing Kiosk Applications*, Journal on Computer Science and Information Systems, 5(1) 41–59, 2008.