Model Highlighting to Improve OCL Learning: A Preliminary Evidence-Based Assessment

***Abstract*—Several authors have claimed that learning OCL is difficult. Based on data gathered from a large number of undergraduate students in the context of two Software Engineering courses, on past school years, we analyze how learning design by contract clauses with UML+OCL compares with several other Software Engineering Body Of Knowledge (SWEBOK) topics. The outcome of the learning process was collected in a rigorous setup, supported by an e-learning platform. We performed inferential statistics on that data to support our conclusions, to identify the relevant explanatory variables for students success/failure.**

Besides aiming at shedding some light on the factors that influence the OCL learning process, we propose a tool-based learning feature, dubbed "model highlighting", reeified as a plugin to Bremen's USE tool. This plugin highlights how an OCL clause traverses a UML class diagram. We performed a preliminary validation of this model highlighting feature by combining an action-research observation period on more than two hundred students in lab sessions in the current school year, with semi-structured interviews, whose conclusions were consolidated by a focus group. The panel of experts unanimously recognized that this model highlighting feature had a positive effect in the OCL learning process.

***Index Terms*—OCL, UML, learning, model highlighting**

I. Introduction

Formal methods is a long discussed topic in Software Engineering, but albeit there are still disputes about under what circumstances formal methods and languages should be used, there appears to be a consensus that formal methods are a difficult topic to learn [1]. One area where formal methods have been employed in a beneficial way, is on providing precision to UML modeling [2]. The Unified Modeling Language (UML) had its first specification in the late nineties and is currently the standard language used in software development for specifying, visualizing, constructing, and documenting software artifacts. However, UML graphical modeling constructs are not precise enough to express all relevant aspects of a specification. To mitigate this problem, a semi-formal language named OCL (Object Constraint Language) [3] was included in the UML standard. OCL is based on first-order logic (predicate calculus) and set theory, provides many collection operations and can be used in several contexts, such as for specifying:

- well-formedness rules or metrics at the metamodel level;
- design by contract clauses (class invariants, pre and post conditions on operations) and derivation rules for attributes or associations in Class Diagrams;
- object queries equivalent to SQL queries in relational databases in Object Diagrams;
- guards on state transitions in State Diagrams.

In this paper we will focus our attention on understanding how humans process a class diagram enriched with OCL clauses, that are built by navigating across the diagram's modeling constructs. UML class diagrams allow describing the structure of a software system in terms of its classes, their attributes and operations, and relationships among classes. This type of UML diagram was recognized, more than one decade ago, as the most widely used one [4] and, according to Google Trends[1], continues to be the one that raises more attention (see Figure 1).

Although OCL can only be claimed to be "semi-formal", since it lacks inference mechanisms (second-order logic), there are conflicting claims (often lacking empirical evidence), on the difficulty of learning OCL, as described in Section II.

In this paper we aim at shedding some light on the factors that influence the OCL learning process, based on the experience of one of this paper's authors, of teaching OCL to undergraduate students for the past several years. For this purpose we contrasted the learning outcomes of a considerable number of students on learning OCL versus a set of several other topics making part of the SWEBOK (Software Engineering Body of Knowledge).

We also report report our recent effort in softening the OCL learning curve, by reducing the psychological and physiological effort needed to correctly interpret an OCL expression and thereby reducing the time required to produce an OCL clause from a natural language specification, in the context of a UML Class Diagram. For this aim we developed a plugin for the USE (UML-based Specification Environment) tool [5] that provides syntax highlighting of OCL expression.

This paper is organized as follows. Section II presents an overview of relevant related work. Section III presents evidence on the difficulty of learning OCL. Section IV describes our implementation of the plugin, from a broader perspective to a more detailed one, as well as a preliminary validation of the functionality, usability, and usefulness from the perspective of experienced users. Section V contains the concluding remarks and addresses some open issues and future research work directions.

II. Related Work

A. UML Diagrams Comprehension

The style and rigor used while producing UML diagrams may vary widely [6]. Several aspects have been pointed out to influence UML models' comprehension. For instance,

[1] https://trends.google.com

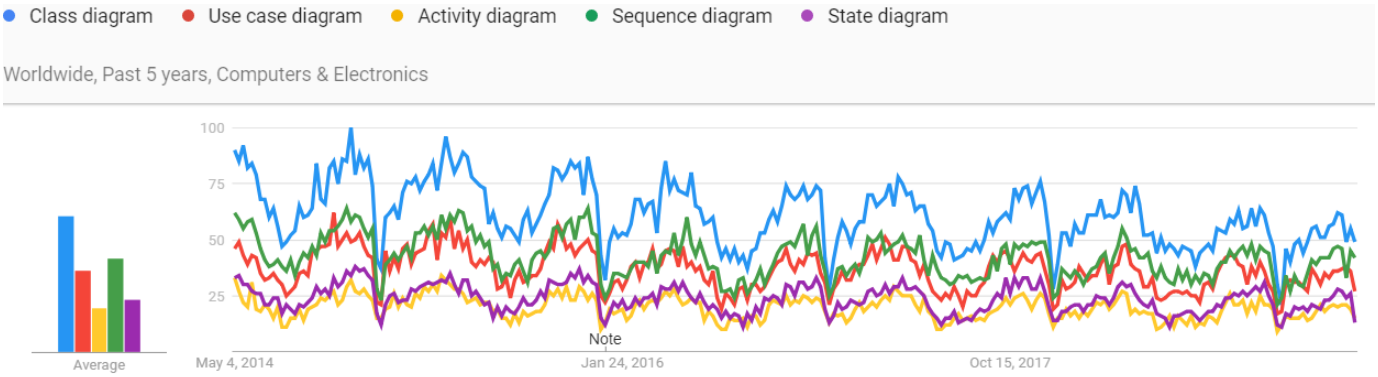

Fig. 1. Percentage of Google searches per UML diagram type, for the last 5 years (up to May 1st, 2019)

empirical evidence was obtained that the level of detail used has a significant effect in UML models comprehension [7].

Guehéneuc [8] and Yusuf [9] used eye-tracking technology to obtain data on how subjects process UML class diagrams, namely how they explore, visualize and examine data. There are three different types of information that are particularly relevant when studying the behavior of the human eye: fixations, saccades, and scanpaths. Fixations are defined by a period of time where the subject maintains the visual gaze on a single location, represented by a pair of coordinates; saccades involve a quick movement between two fixation points; scanpaths are defined as directed paths formed by saccades between fixations. Results of this study were used to create gaze plots of UML class diagrams with fixations, saccades, and scanpaths, and heatmaps of fixations, where it is possible to discern the path taken by the subject's eyes through the diagram and which were the main focus points. This information is particularly important when trying to understand how subjects visually process class diagrams. Relevant conclusions show that relationships mainly contain fixations on its ends and saccades on the line, indicating that lines are mainly used for navigation purposes. A large number of fixations on a *stimulus*, an object that is viewed by a subject, indicates a greater effort to understand the object in question, due either to its complexity or to a poor layout. Coding information using color was found useful on subjects with a greater level of expertise when trying to explore class diagrams more efficiently.

B. OCL Comprehension

Several studies have been conducted to assess the benefits of using OCL alongside UML models [10] [11], which has been proven advantageous to modelers, once they overcome OCL's learning curve. It has been claimed that learning OCL is not recommended for UML novices [12]. It has also been claimed that for assessing details of a model, OCL may lead to complex expressions that increase its learning curve [13]. However, other authors observed that *"participants working with OCL performed consistently well on specifying constraints of varying complexity suggesting that OCL can be used to model complicated constraints (commonly observed in industrial applications) with the same quality as for simpler constraints"* [14].

The impact of OCL in UML-based development has been discussed by several researchers, focusing on its influence on comprehension and maintainability of UML models, and whereas the additional effort and formality justify the benefit. Briand et al [10] [11] investigated the impact of OCL during development, understanding, and modification of UML analysis documents, which are three typical software engineering activities. The authors compared subjects (4^{th} year software/-computer engineering students) performance on comprehension and maintenance tasks when using OCL to document invariants, pre- and post-conditions, which are typically documented in natural language. Results showed that OCL has the potential to promote comprehension and modification of UML diagrams, but requires a high level of experience and training.

It is plausible that the cognitive effort needed to comprehend, design, and/or maintain UML models is influenced by the levels of complexity that characterize OCL expressions. A set of OCL complexity metrics was proposed in [15] to explain different levels of comprehensibility and maintainability. Reynoso et al [16] [17] pursued their effort of understanding the effects due to OCL expressions' complexity by applying techniques such as chunking (which involves recognizing groups of declarations and the extraction of information, that is remembered as a chunk) and tracing (which involves scanning through a program in different directions in search of relevant chunks). While there are many distinct metrics that capture different aspects that affect expression's complexity, these authors decided to focus on the impact of tracing-based metrics, because this technique was proven as an essential activity in program comprehension [15]. They found statistical significance on the impact of tracing-related metrics on the understandability and, consequently, modifiability of OCL expressions. Their research rationale is shown in Figure 2: the structural properties of an OCL expression affect the cognitive complexity needed to comprehend that expression, which consequently influences understandability and maintainability. Table I shows the metrics in question and their classifications, depending on their focus on the structural properties of OCL

expressions: coupling (degree of interdependence between objects), size, and length.

Fig. 2. Relationship between structural properties of an OCL expression, cognitive complexity related to tracing, understandability and maintainability (based on [15])

TABLE I
TRACING-RELATED OCL EXPRESSION METRICS (BASED ON [15])

Acronym	Description	Classification
NNR	Number of Navigated Relationships	C
NAN	Attributes referred through Navigation	C
WNO	Weighted Number of referred Operations through Navigations	C
NNC	Number of Navigated Classes	C
WNM	Weighted Number of Messages	C, S
NPT	Number of Parameters whose Types are classes defined in a class diagram	C
NUCA	Number of Utility Classes Attributes Used	C
NUCO	Number of Utility Classes Operations Used	C
DN	Depth of Navigations	L
WNN	Weighted Number of Navigations	S
WCO	Weighted Number of Collection Operations	S

Note: In the table above, C stands for Coupling, S stands for Size, and L stands for Length.

For containment sake, we reproduce the definition of this OCL metrics set (taken from [15]), since we will use it in our analysis in Section III:

Definition 1. NNR Metric (Number of Navigated Relationships): Measures the total number of navigated relationships in an expression. Relationships are only counted once, even if they're navigated multiple times.

Definition 2. NAN Metric (Number of Attributes referred through Navigations): Counts the number of attributes referred through navigations.

Definition 3. WNO Metric (Weighted Number of referred Operations through navigations): sum of weighted operations (where the weight is defined by the number of *in/out* parameters, including the return type) mentioned through navigations.

Definition 4. NNC Metric (Number of Navigated Classes): number of classes, association classes or interfaces referred through navigations.

Definition 5. WNM (Weighted Number of Messages): sum of weighted messages (where the weight is defined by the number of parameters) present in an expression.

Definition 6. NPT (Number of Parameters whose Types are classes defined in the Class Diagram): number of different classes and interfaces used as *in/out* parameters or result.

Definition 7. NUCA (Number of Utility Class Attributes used): number of utility classes' attributes used in an expression; attributes that belong to the same utility class are only counted once, even if they are referred multiple times.

Definition 8. NUCO (Number of Utility Class Operations used): similar to the NUCA metric, but instead of attributes it considers operations.

Definition 9. DN (Depth of Navigations): maximum depth of a navigation tree, where the root node represents the class name of the contextual instance (*self*); for each navigation that starts from *self*, we create a branch that connects to a new node, which represents the navigated class; a new tree is created if a navigation contains a collection operation defined in terms of new navigation(s), and is connected to the original tree through a "definition connection".

Definition 10. WNN (Weighted Number of Navigations): sum of weighted navigations presented in an expression, where the weight is defined by the level on which the operation is used in an expression.

Definition 11. WCO (Weighted Number of Collection Operations): sum of weighted collection operations, where the weight is defined by the level on which the operation is used in an expression.

C. Impact of Syntax Highlighting on Software Comprehension

Understandability (aka comprehensibility) has long been recognized as one of the main facets of software systems' maintainability [18]. Software comprehension can be affected by many features, including naming scheme, indentation, commenting, and documentation. There is also empirical evidence that syntax highlighting, a feature present in some text editors that allow displaying text in different colors (including background color) and fonts depending on its categorization, significantly improves software comprehension.

Initial studies were mainly concerned with the effect of colors on program comprehension in order to facilitate the learning process and increase developers' productivity, as the majority of tools was only available in black and white. Rambally [19] studied the effect of two color schemes on program comprehension when compared to a version of the same program presented in black and white: Color-scheme-A used different colors for code blocks, e.g. loops and conditionals; Color-scheme-B used different colors to identify statements and functions, e.g. I/O (Input/Output), decision and declaration statements, and variable binding. From a sample of 44 intermediate-level (novice) and 35 senior-level (expert) programming students, results showed that, in general, both groups can better understand programs when using Color-scheme-B. An interesting note mentioned in this experiment is that participants subjectively favored Color-scheme-A as the easiest to understand, even tough results supported Color-scheme-B as the most effective for the completion of the given comprehension tasks.

Several years after Rambally's study, Sarkar [20] did an experiment to evaluate the impact of syntax coloring on program comprehension time, using 10 graduate computer

science students. Even though the experiment's validity is threatened by the small number of participants, it showed that assignments were solved remarkably faster when using syntax highlighting. The author collected the participants' sessions with an eye-tracking device and discerned that syntax highlighting directed participants' focus on to smaller regions of the code, and, in some cases, even some keywords were completely ignored. Another interesting remark taken from this investigation is that the benefits of syntax highlighting in novice programmers are significantly more relevant when compared to experienced programmers, which might indicate that it's a useful feature when learning a certain language, but less important for using it.

Although much less than in code, model highlighting has also been long used to improve comprehension of UML models. In [21], color is used to indicate groups of classes belonging to the same design pattern. Later, in [22], a search and highlight feature was proposed to help identify sections of interest in large UML models.

An important aspect when using highlighting is to decide which features to include, and how to code them. It is essential to determine which is the most efficient way to display certain information, e.g. decide to use color X over Y for a specific category of information, decide to use a background color instead of text color, etc. Mehta and Zhu [23] focused their attention on the effect of colors (red and blue) on cognitive task performance. They started by describing the theory defended by color theorists which states that people create specific associations to colors, depending on the repeated situations they experience with that color: red is an intensive color normally associated with errors or dangers, while blue is mostly associated with calmness, peace, and tranquility. Results from this research show that distinct colors offer different benefits, and the choice of the right color depends on the nature of the task. As red primarily activates an avoidance motivation, it mainly enhances performance on detail-oriented tasks. On the other hand, blue activates an approach motivation which is more beneficial for creative tasks.

Several support tools have emerged in the last two decades to assist model-driven development, including the analysis and design phases, where modelers need to interpret and write OCL expressions [24]. These tools provide a variety of useful functionalities including syntactic analysis, traceability to the UML model, and debugging.

However, to the best of our knowledge, none of these tools provides "OCL to model highlighting". Our hypothesis is based on previous studies [19] [9] [23] [20] that investigated the impact of visual aspects (color, layout, font, ...) on program comprehension, revealing positive impacts on subject's comprehensibility when enhancing programs with visual features.

III. On OCL Complexity

A. Evidence on the Relative Difficulty of Learning OCL

In this section, we present evidence on the complexity of learning OCL, when compared to a set of other Software Engineering topics offered in two courses that together span a full school year. The courses' syllabuses were organized according to the following areas of the Software Engineering Body Of Knowledge (SWEBOK): Software Requirements, Software Design, Software Construction, Software Testing, Software Maintenance, Software Configuration Management, Software Engineering Management, Software Engineering Process, Software Engineering Tools and Methods, Software Quality. Although OCL was taught as part of the Software Design area, it was considered separately in this study. In the Software Design area were kept other topics, namely the one of "design patterns".

After taking each area, learning was assessed through comprehensive questionnaires of closed true-false questions, through an e-learning system. Data was collected in two consecutive years, totalling over 150 students enrolled in three computer science related degrees[2].

The OCL questionnaire was constructed as follows, to guarantee a true-false answer: the same UML class model with no more than 20 classes (to fit legibly in a computer screen) was made available to all students in advance and its semantics explained in detail. A large set of objects and links is given to students right before they start answering the questionnaire. Students instantiate the class model using the USE tool [5] and can use it freely while answering the questionnaire. Each of its natural language questions could be answered by formulating a quantitative OCL upon the instantiated model. For instance, the correct answer to *"how many tickets were sold for the flights from Denmark to the USA, during last March?"* would be 245. To be considered a true answer, students had to fill that value in the e-learning platform.

Figure 3 presents some descriptive statistics on the grades obtained in OCL and in the topics framed by the SWEBOK knowledge areas. Notice that the topics in the area of *Software Requirements* were only taken by students of 1 of the 3 degrees, while the topics of *Software Construction* were taken by 2 of the 3 degrees.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
OCL	156	0,00%	100,00%	50,9179%	33,50341%
Software Engineering Management	124	0,00%	93,33%	54,2314%	22,06163%
Software Quality	159	0,00%	94,44%	54,3902%	21,03014%
Software Requirements	59	5,00%	100,00%	58,1185%	21,84444%
Software Testing	141	0,00%	100,00%	60,4914%	22,06326%
Software Configuration Management	133	0,00%	100,00%	61,8988%	19,04061%
Software Engineering Process	132	0,00%	95,00%	65,1486%	14,14017%
Software Design	121	0,00%	100,00%	65,7350%	22,70123%
Software Construction	103	14,29%	100,00%	67,7011%	21,92754%

Fig. 3. Descriptive Statistics for the Learning Grades

OCL was the topic obtaining the worst grade, on average, but it is important to check if the distribution of grades in OCL is significantly different from the ones obtained in the other

[2]The collected dataset used in this section is made available at http://XXXX (obscured to guarantee MODEL's double-blind review process)

topics. We started by performing a One-Sample Kolmogorov-Smirnov Test, with Lilliefors significance correction, that showed that the grades obtained in most of the topics/areas were not normally distributed. Therefore, we applied the non-parametric Related-Samples Wilcoxon Signed Rank Test between the grades obtained in OCL and the ones obtained for each SWEBOK area separately. The results are shown in Table II for a significance level of 0.05.

TABLE II
RELATED-SAMPLES WILCOXON SIGNED RANK TEST RESULTS
(LEARNING GRADES IN OCL VERSUS IN OTHER SWEBOK TOPICS)

SWEBOK Area	Test Sign.	Decision
Software Requirements	.719	Retain null hypot.
Software Design	.000	Reject null hypot.
Software Construction	.000	Reject null hypot.
Software Testing	.011	Reject null hypot.
Software Maintenance	.026	Reject null hypot.
Software Configuration Management	.000	Reject null hypot.
Software Engineering Management	.187	Retain null hypot.
Software Engineering Process	.000	Reject null hypot.
Software Engineering Tools and Methods	.006	Reject null hypot.
Software Quality	.487	Retain null hypot.

A Friedman ANOVA test confirmed that there are no significant differences in the distribution of grades obtained in OCL and in the SWEBOK topics of Software Requirements, Software Quality and Software Engineering Management as represented in Figure 4.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.	Decision
1	The distributions of OCL, Software Quality, Software Requirements and Software Engineering Management are the same.	Related-Samples Friedman's Two-Way Analysis of Variance by Ranks	,182	Retain the null hypothesis.

Asymptotic significances are displayed. The significance level is ,05.

Fig. 4. Related Samples Friedman ANOVA Test

B. Which factors determine the success in learning OCL?

In this section, we present the analysis of the influence of OCL complexity on the success rate of student's assessments, as well as the influence of Natural Language complexity on the same outcome. Students underwent a test to assess their competencies in OCL, as described earlier. An example of a possible question and respective OCL expression can be seen on Expression 1.

For the first case study, we are interested in assessing if OCL complexity can explain the learning success rate, and which metrics (from the ones described in section II-B) are relevant to explain that success. Since many metrics were proposed in the literature we must produce a variable reduction, not to obtain an overspecified model. We started by assessing the normality of the given metrics, using a Kolmogorov-Smirnov test. The results in Figure 5 indicates that the proposed OCL metrics do not follow a normal distribution, D(140) = .19, p = .00 (for NNR metric, other metrics show similar results).

One-Sample Kolmogorov-Smirnov Test

	N	Normal Parameters[a,b] Mean	Std. Deviation	Most Extreme Differences Absolute	Positive	Negative	Test Statistic	Asymp. Sig. (2-tailed)
NNR	140	2,40	1,516	,190	,190	-,140	,190	,000[c]
NAN	140	,74	,903	,300	,300	-,207	,300	,000[c]
WNO	140	23,79	30,527	,258	,258	-,218	,258	,000[c]
NNC	140	3,59	1,569	,153	,153	-,145	,153	,000[c]
NUCA	140	,16	,626	,528	,528	-,401	,528	,000[c]
NUCO	140	,04	,203	,541	,541	-,417	,541	,000[c]
WNN	140	10,33	8,504	,112	,112	-,112	,112	,000[c]
DN	140	8,91	6,037	,110	,110	-,073	,110	,000[c]
WCO	140	12,74	15,761	,240	,240	-,228	,240	,000[c]

a. Test distribution is Normal.
b. Calculated from data.
c. Lilliefors Significance Correction.

Fig. 5. Kolmogorov–Smirnov test on the normal distribution of OCL Complexity metrics

To evaluate if the complexity of OCL expression (answers) can explain the success of students answers, and because the independent variables (metrics) don't follow a normal distribution, we applied a Spearman's rho correlation coefficient to evaluate their correlation, and their effect on the dependent variable (student's success). Figure 6 presents the result of this assessment. We decided to exclude metrics with strong correlation (painted with red), as they can be used to explain the same results, and keep the ones that are more dissimilar. The resulting metrics were NAN, WNO, NUCO, and WCO, as they revealed greater influence on the dependent variable.

Spearman's rho correlation coefficient

	NNR	NAN	WNO	NNC	NUCA	NUCO	WNN	DN	WCO	Average Success
NNR	1,000	-,389**	-,332**	,937**	-0,099	-0,056	,832**	,830**	,493**	-0,011
NAN	-,389**	1,000	,668**	-,313**	,172*	0,132	-,357**	-,253**	-0,108	-0,130
WNO	-,332**	,668**	1,000	-,259**	,415**	,358**	-,285**	-,194*	-,208*	-,242**
NNC	,937**	-,313**	-,259**	1,000	-0,140	-0,093	,849**	,844**	,546**	-0,034
NUCA	-0,099	,172*	,415**	-0,140	1,000	,786**	-0,114	-0,026	-,196*	-0,013
NUCO	-0,056	0,132	,358**	-0,093	,786**	1,000	-0,138	-0,037	-,240**	0,106
WNN	,832**	-,357**	-,285**	,849**	-0,114	-0,138	1,000	,955**	,789**	-0,066
DN	,830**	-,253**	-,194*	,844**	-0,026	-0,037	,955**	1,000	,742**	-0,045
WCO	,493**	-0,108	-,208*	,546**	-,196*	-,240**	,789**	,742**	1,000	-,179*
Average Success	-0,011	-0,130	-,242**	-0,034	-0,013	0,106	-0,066	-0,045	-,179*	1,000

**. Correlation is significant at the 0.01 level (2-tailed).
*. Correlation is significant at the 0.05 level (2-tailed).

Fig. 6. Spearman's rho correlation coefficient of OCL Complexity metrics

A similar study was applied to Natural Language metrics [25], which were calculated for each question. As many metrics were proposed, we must assemble a variable reduction, again to prevent an overspecific model. We started by assessing the normality of the given metrics, using a Kolmogorov-Smirnov test. The results in Figure 7 indicates that the majority of the proposed Natural Language metrics do not follow a normal distribution, D(140) = .089, p = .009 (for Flesch–Kincaid Grade Level metric, other metrics show similar results).

To evaluate if the complexity of Natural Language (questions) can explain the success of students answers, and because the independent variables (metrics) don't follow a normal distribution, we applied a Spearman's rho correlation coefficient to evaluate their correlation, and their effect on the dependent variable (student's success). Figure 8 presents the result of

One-Sample Kolmogorov-Smirnov Test

	N	Normal Parameters[a,b] Mean	Std. Deviation	Most Extreme Differences Absolute	Positive	Negative	Test Statistic	Asymp. Sig. (2-tailed)
fleschKincaidGradeLevel	140	7,440	2,9646	,089	,089	-,054	,089	,009[c]
gunningFogIndex	140	8,661	3,6010	,163	,146	-,163	,163	,000[c]
colemanLiauIndex	140	8,614	3,1360	,072	,072	-,042	,072	,075[c]
smogIndex	140	8,581	3,5931	,281	,179	-,281	,281	,000[c]
automatedReadibilityIndex	140	6,538	3,1660	,069	,069	-,033	,069	,200[c,d]
forecastGradeLevel	140	10,739	1,7896	,075	,075	-,059	,075	,055[c]
powersSumnerKearlScore	140	5,064	,8637	,180	,180	-,160	,180	,000[c]
rix_score	140	7,100	2,3728	,198	,145	-,198	,198	,000[c]
fleschReadingEase	140	64,122	18,4639	,077	,044	-,077	,077	,042[c]
spacheScore	140	4,667	1,0490	,116	,078	-,116	,116	,000[c]
newDaleChallScore	140	3,980	2,0622	,105	,105	-,095	,105	,001[c]
lix_score	140	35,487	11,3251	,103	,103	-,099	,103	,001[c]
lensear_write	140	77,950	14,1087	,091	,085	-,091	,091	,006[c]

a. Test distribution is Normal.

b. Calculated from data.

c. Lilliefors Significance Correction.

d. This is a lower bound of the true significance.

Fig. 7. Kolmogorov–Smirnov test on the normal distribution of Natural Language metrics (English)

this assessment. We decided to exclude metrics with strong correlation (painted with red), as they can be used to explain the same results, and keep the ones that are more dissimilar. The resulting metrics were the Automated Readability Index, Forest Grade Level, New Dale-Chall, and Lensear Write, as they revealed greater influence on the dependent variable.

Spearman's rho correlation coefficient

	fleschKincaidGradeLevel	gunningFogIndex	colemanLiauIndex	smogIndex	automatedReadibilityIndex	forecastGradeLevel	powersSumnerKearlScore	rix_score	fleschReadingEase	spacheScore	newDaleChallScore	lix_score	lensear_write	Average Success
fleschKincaidGradeLevel	1,000	,841**	,678**	,858**	,802**	,505**	,828**	,736**	-,928**	,476**	,565**	,740**	-,715**	-,197*
gunningFogIndex	,841**	1,000	,566**	,878**	,729**	,258**	,976**	,659**	-,739**	,369**	,441**	,654**	-,515**	-,217*
colemanLiauIndex	,678**	,566**	1,000	,617**	,874**	,529**	,560**	,709**	-,749**	0,132	,285**	,813**	-,450**	-,234**
smogIndex	,858**	,878**	,617**	1,000	,677**	,368**	,866**	,698**	-,822**	,416**	,545**	,688**	-,543**	-0,113
automatedReadibilityIndex	,802**	,729**	,874**	,677**	1,000	,319**	,694**	,820**	-,704**	,285**	,289**	,837**	-,594**	-,336**
forecastGradeLevel	,505**	,258**	,529**	,388**	,319**	1,000	,284**	,285**	-,707**	,271**	,453**	,444**	-,411**	0,047
powersSumnerKearlScore	,828**	,976**	,560**	,866**	,694**	,284**	1,000	,619**	-,757**	,377**	,483**	,635**	-,476**	-,173*
rix_score	,736**	,659**	,709**	,698**	,820**	,285**	,619**	1,000	-,630**	,407**	,376**	,934**	-,653**	-,234**
fleschReadingEase	-,928**	-,739**	-,749**	-,822**	-,704**	-,707**	-,757**	-,630**	1,000	-,391**	-,578**	-,719**	,605**	0,105
spacheScore	,476**	,369**	0,132	,416**	,285**	,271**	,377**	,407**	-,391**	1,000	,753**	,364**	-,506**	0,060
newDaleChallScore	,565**	,441**	,285**	,545**	,289**	,453**	,483**	,376**	-,578**	,753**	1,000	,409**	-,486**	0,150
lix_score	,740**	,654**	,813**	,688**	,837**	,444**	,635**	,934**	-,719**	,364**	,409**	1,000	-,578**	-,206*
lensear_write	-,715**	-,515**	-,450**	-,543**	-,594**	-,411**	-,476**	-,653**	,605**	-,506**	-,486**	-,578**	1,000	0,086
Average Success	-,197*	-,217*	-,234**	-0,113	-,336**	0,047	-,173*	-,234**	0,105	0,060	0,150	-,206*	0,086	1,000

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Fig. 8. Spearman's rho correlation coefficient of NL metrics (English)

Finally, after reducing the OCL and Natural Language metrics, we performed a Linear Regression to assess the capability of those metrics to explain the success of students, as Spearman's rho can evaluate relative monotonies whether the models are linear or not.

The results of the regression on OCL metrics are shown in Figure 9. A significant regression equation was found F(4, 135) = 4.313, $p < .01$, with an R^2 of .113.

The results of the regression on Natural Language metrics are shown in Figure 10. A significant regression equation was found F(4, 135) = 6.311, $p < .001$, with an R^2 of .158.

The resulting linear model in both cases (OCL and Natural Language metrics) revealed a poor explanatory power on the dependent variable, meaning that there is a monotony relationship, but it is not linear.

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics R Square Change	F Change	df1	df2	Sig. F Change
1	,337[a]	,113	,087	,20126249	,113	4,313	4	135	,003

a. Predictors: (Constant), WCO, WNO, NAN, NUCO

ANOVA[a]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	,699	4	,175	4,313	,003[b]
	Residual	5,468	135	,041		
	Total	6,167	139			

a. Dependent Variable: Average Success

b. Predictors: (Constant), WCO, WNO, NAN, NUCO

Fig. 9. Linear Regression using NAN, WNO, NUCO, and WCO to explain the success

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics R Square Change	F Change	df1	df2	Sig. F Change
1	,397[a]	,158	,133	,19618009	,158	6,311	4	135	,000

a. Predictors: (Constant), lensear_write, forecastGradeLevel, automatedReadibilityIndex, newDaleChallScore

ANOVA[a]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	,972	4	,243	6,311	,000[b]
	Residual	5,196	135	,038		
	Total	6,167	139			

a. Dependent Variable: Average Success

b. Predictors: (Constant), lensear_write, forecastGradeLevel, automatedReadibilityIndex, newDaleChallScore

Fig. 10. Linear Regression using Automated Readability Index, Forest Grade Level, New Dale-Chall, and Lensear Write to explain the success

IV. A Plugin for Improving OCL Learning

A. The USE Tool

Currently, there are several UML tools supporting OCL. Here we focus on USE (UML-based Specification Environment), since it is the tool that we use for teaching purposes. USE is one of the pioneering OCL tools, initially developed in Java by Mark Richters as a PhD dissertation project at the University of Bremen [26]. It is freely distributed under GPLv2 (GNU General Public License version 2.0), and the latest version (5.1.0), was released in April 2019. Main functionalities include syntactic analysis, type checking, dynamic validation of invariants and pre- and post-conditions, and consistency checking. Class diagrams with UML constraints are specified in plain text in file with *".use"* extension. USE provides an evaluation browser (see Figure 11) that allows users to manually introduce OCL expressions and debug them, as it generates an abstract syntax tree (AST) from the input text that is evaluated against the model and returns either a syntax error or the result of the given expression. Last, but not the least, USE has an highlighting feature to represent invariants' coverage. This feature was very helpful since it provided primitives for the development of our plugin.

B. The OCL Highlight Plugin

Aside from the functionalities provided by USE described in the previous section, this tool allows third parties to provide additional functionalities through plugins. In this section, we describe our implementation of the OCL Highlight Plugin, which was developed in Java [28]. This plugin provides a new OCL evaluation dialog, which resembles the one already

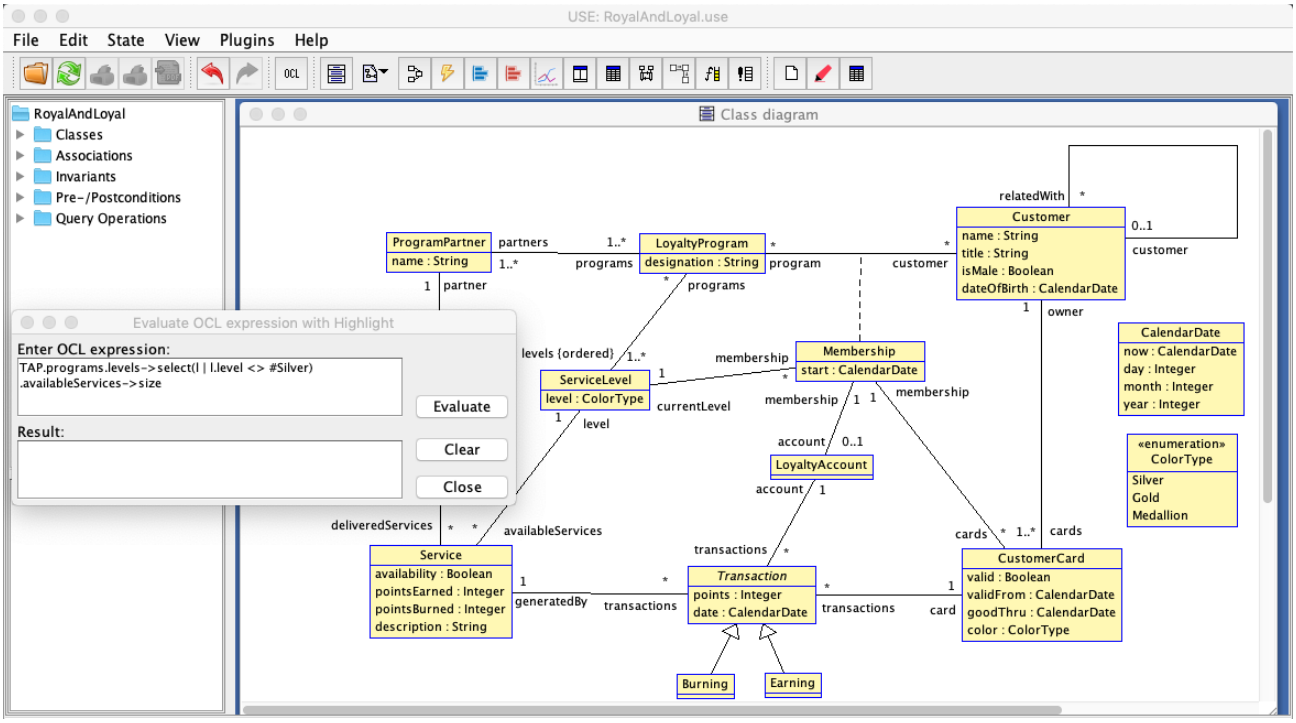

Fig. 11. Evaluation browser on USE (Royal and Loyal example from [27])

provided by the tool, that offers syntax highlighting in the UML model when users evaluate a given OCL expression.

To make use of this plugin, users should download the corresponding *.jar* file and place it in the *plugins* folder of USE installation. After restarting USE, a new button with a red marker's icon will be available. After importing a *.use* file with the definition of the model and a *.soil* file with the instantiation of the respective model (objects), the user should create a Class Diagram view. When clicking on the *Highlight Expression* button, an evaluation browser will open. In this browser, the user can input OCL expressions, and after clicking on *Evaluate*, assuming that the expression is syntatically correct, the elements that are referenced in the expression (classes, properties, and associations) become highlighted in the class diagram, while the remaining elements become deemphasized. Expression 1 and Expression 2 illustrate the syntax highlighting provided by the evaluation dialog, using the Royal and Loyal model [27].

Expression 1: This expression exemplifies how to get the balance of points of TAP, which is an instance of *ProgramPartner*. The correspondent highlight is illustrated in Figure 12. The highlighted element are: *ProgramPartner* class, which represents TAP; the navigation from *ProgramPartner* to *Service* (rolename *deliveredServices*), and the *Service* class; and the navigation from *Service* to *Transaction* (rolename *transaction*), and the class *Transaction*. In this expression we want to collect the sum of *points* (property) of a *Transaction*. If a *Transaction* is of type *Earning*, the points are counted positively. If not, they are counted negatively. Both *Earning* and *points* of *Transaction* are also highlighted.

```
TAP.deliveredServices.transactions
    ->collect(t | if(t.oclIsTypeOf(Earning))
        then t.points else -t.points endif)
    ->sum
```

Expression 1. OCL expression 1

Expression 2: This expression exemplifies how to get the number of non Silver services provided by TAP in the participating Loyalty Programs. The correspondent highlight is illustrated in Figure 13. The highlighted elements are: *ProgramParner*, which again represents TAP; the navigation between *ProgramPartner* and *LoyaltyProgram* (rolename *programs*), and *LoyaltyProgram* class; the navigation between *LoyaltyProgram* and *ServiceLevel* (rolename *levels*), and the class *ServiceLevel*; the *level* (property) of the class *ServiceLevel* and its corresponding type (*ColorType* enum); the navigation from *ServiceLevel* to *Service* (rolename *availableServices*), and the class *Service*.

```
TAP.programs.levels
    ->select(l | l.level <> #Silver)
    .availableServices->size
```

Expression 2. OCL expression 2

As mentioned above, this plugin provides a dialog similar to the already existing evaluation dialog, with the additional functionality of syntax highlighting. This dialog uses the same code as the original, but not only it allows the evaluation of the expressions, it also processes them using a Visitor pattern [29] for OCL expressions, that is invoked after their evaluation. The implemented Visitor inherits from an interface available in the

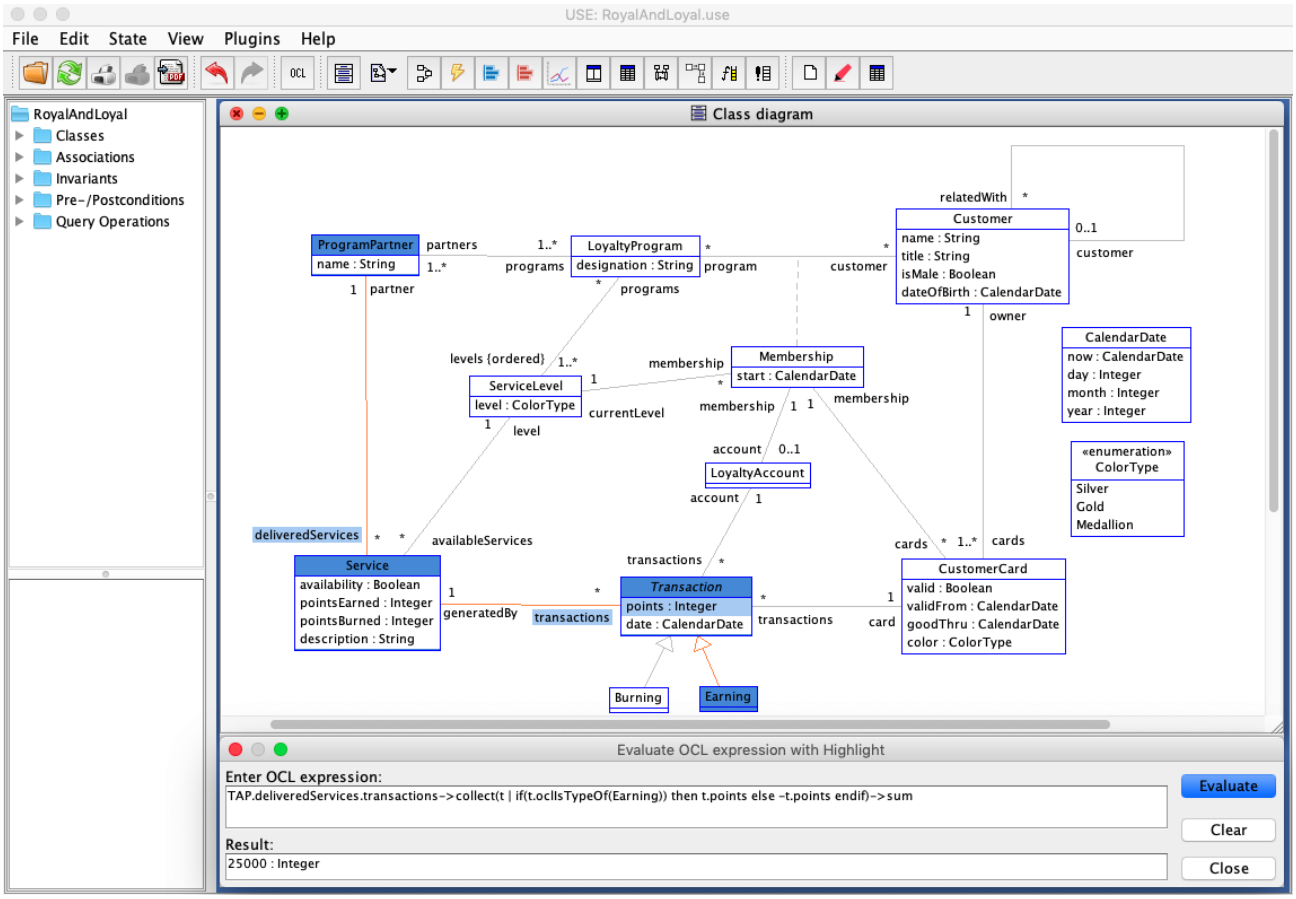

Fig. 12. OCL Highlight Plugin: Expression 1

original tool and enables highlighting the different elements of the UML class diagram. Since the API provided by the original tool did not always provide methods to customize the components of the class diagram, it was crucial to make the necessary changes using reflection.

C. Plugin Validation

In this subsection, we present the focus group study we conducted to obtain feedback and experiences from UML/OCL experts (teachers) when using the OCL Highlight plugin to teach OCL to students at the university. This experience followed the available guidelines on how to plan and run focus groups [30].

1) Defining the research problem.: The objective of the study was to provide insights into the utility of the developed plugin to ease the learning of OCL from the perspective of UML/OCL experts (teachers). Furthermore, we also wanted to obtain suggestions for improvement of the plugin and teachers opinions on how students reacted to the plugin.

2) Selecting the participants.: Six teachers from the Department of Computer Science at XXXXXXXXXXXX[3] were invited by e-mail to take part in the focus group discussions. The selected invitees are Software Engineering experts and are used to teach UML with OCL to bachelor students.

3) Planning and conducting the focus group session.: We designed the focus group session to consist of two parts. On the first part, teachers were asked to individually observe their respective students during a week of classes, where students were submitted to their first contact with USE tool. At the end of the week, the teachers were invited to attend a one-hour meeting to discuss their observations and fill out a previously prepared short questionnaire about the usefulness of the plugin and the reaction of students to it.

4) Analysis: Results on Table II indicate that UML/OCL experts agreed on the usefulness of the plugin and that it helps students to understand how OCL transverses a class diagram. This assessment was very helpful to show that students can easily operate with the plugin, as we did not want to aggravate the difficulty in learning by introducing another element that they would need to manage. As a last important remark, some teachers also referred that students thought that the highlighting was something obvious, which again might indicate that it provides a natural integration with the tool.

[3](obscured to guarantee MODEL's double-blind review process)

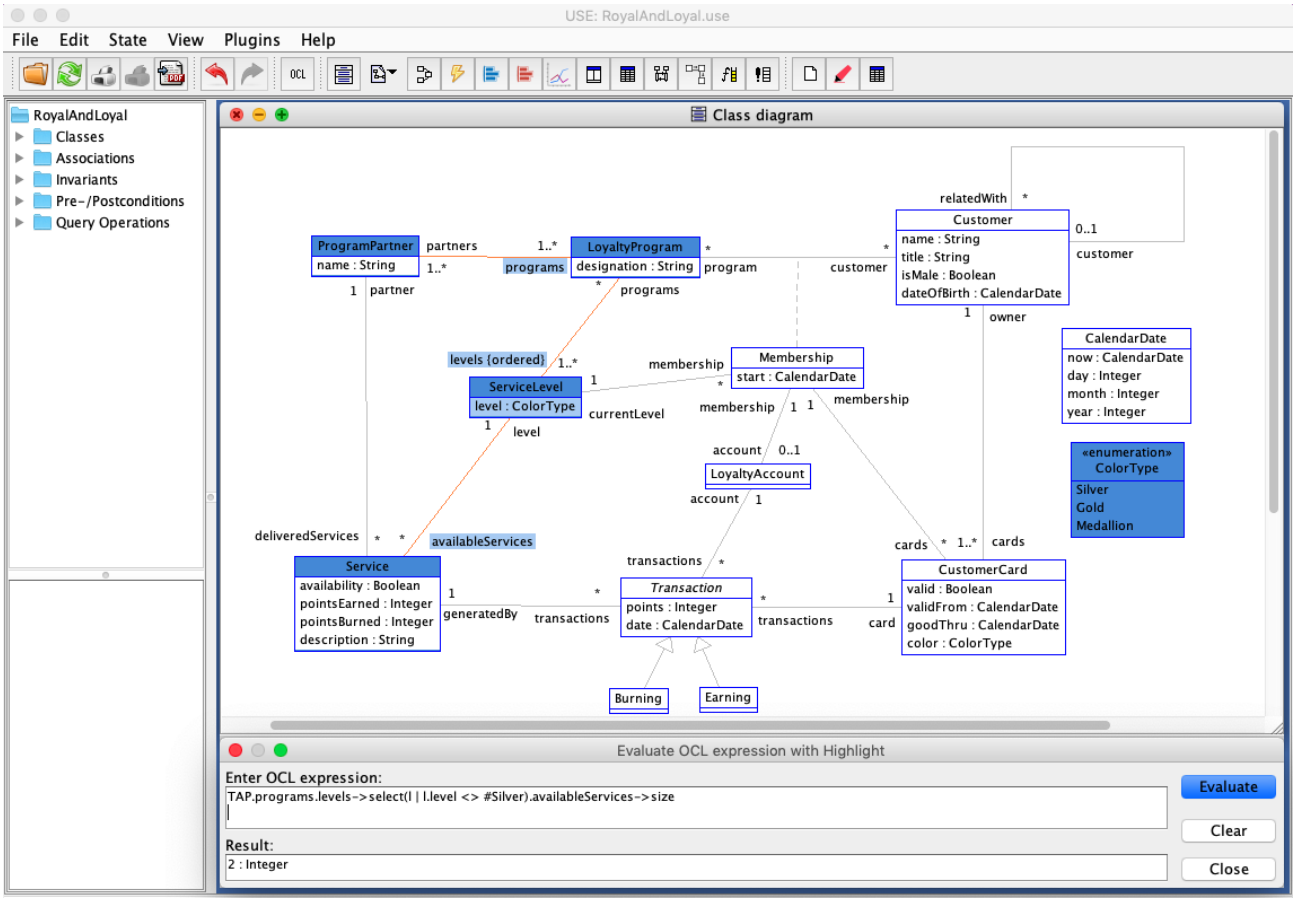

Fig. 13. OCL Highlight Plugin: Expression 2

V. Conclusions and Future Work

A. Conclusions

We collected data systematically on the outcome of the learning process of a set of SWEBOK topics, from a large set of students, with the help of an e-learning platform, through extensive closed questions questionnaires. OCL did not clearly emerge as the most difficult topic.

We also performed some inferential statistics to identify the cause of students success/failure in learning OCL, based on questionnaires where students had to translate natural language clauses to OCL. The explanatory variables that we explored were: (i) the linguistic complexity of the questions (problem domain) in natural language, (ii) the complexity of the OCL clauses, as measured by a set of metrics proposed elsewhere, required to answer those questions (solution domain), and (iii) students overall proficiency. We controlled for the origin of students, since they were following two different degrees (computer science and business informatics) and the school year, since although the overall degree and course syllabus have remained constant during the observation period, variations in students background might have occurred. Neither individually, nor in conjunction, were the aforementioned variables able to provide an acceptable explanatory power, as denoted by low Low R-squared values. Besides shedding some additional light on the factors that influence the OCL learning process, we proposed a tool-based learning feature, dubbed "model highlighting", reeified as a plugin to Bremen's USE tool. This plugin highlights how an OCL clause traverses a UML class diagram. We performed a preliminary validation of this model highlighting feature by combining an action-research observation period on more than two hundred students in lab sessions in the current school year, with semi-structured interviews, whose conclusions were consolidated by a focus group. The panel of experts unanimously recognized that this model highlighting feature had a positive effect in the OCL learning process.

B. Future Work

The initial validation of this plugin was beneficial for the detection of bugs, which are planned to be fixed in a future version, along with the inclusion of the most relevant suggestions provided by the experts. Additional experiments are required to corroborate the utility of the plugin, including extensive use by students in evaluation moments. Results of this experiment will indicate if the developed plugin was beneficial, or not, for students to learn OCL. Likewise, we intend to present further evidence on the relative difficulty of learning OCL by collecting and analyzing assessments data

TABLE III
QUALITATIVE VALIDATION OF THE OCL HIGHLIGHT PLUGIN

QUESTIONS	What is your opinion, as a UML/OCL expert, on the usefulness of this plugin? Considering that you have used the tool without the plugin, do you think it can help students when learning OCL?	How did the students in your class(es) react to the plugin? In particular, the assimilation of the visual metaphor was easy, that is the correspondence between the textual expression in OCL and the highlighted elements in the class diagram?
Expert 1	An indispensable and missing plugin, so far! While learning OCL, it is critical to understand the relationship between the often complex OCL expressions and related modeling elements defined in a class diagram – often complex as well. OCL Highlight plugin delivers a simple, yet complete, graphical representation of those relationships, thus facilitating the understanding and learning of OCL expressions.	The students quickly understood the usefulness and ease of use of the plugin. The visual matching between the OCL expression and the related modeling elements in the class diagram was considered very intuitive. The graphical feedback also motivated the students to try out and understand increasingly complex OCL expressions.
Expert 2	I consider that the plugin is of great utility since it illustrates the navigation of the queries in OCL. It facilitates the understanding of the expressions, as well as the understanding of OCL language.	The reaction of the students to the plugin was something natural as if it was something obvious that the tool "painted the way". I took the opportunity of asking them if the path was not painted in the class diagram would be more difficult to understand the semantics of queries, and most students said yes.
Expert 3	The plugin is extremely important as students have some difficulties to understand OCL and, more importantly, to build OCL expressions. This visual insight helps to grasp how OCL operates over the model.	Yes, they have immediately realized which parts of the model were involved.
Expert 4	It is a very useful plugin since it helps the students understand what classes are being used in each OCL expression. It introduces traceability and shows a better insight to the students, making it easier for them to learn.	Yes, they understood quickly how OCL expressions worked.
Expert 5	This plugin is mostly a good facilitator and a really useful tool for the understanding of UML/OCL, mainly when we have class diagrams with a considerable size. Especially for students, it can ease and improve the process of learning OCL queries.	In class and with this plugin, students were able to follow the OCL expressions they were writing by immediately visualizing the result of those queries in terms of the used classes, associations and attributes that were highlighted. Students accepted the use of this tool really well, compared with the previous year, it not only provided a more engaging learning experience for the students but also, as a lecturer, it helped to demonstrate the process of querying.
Expert 6	I believe that, in fact, this plugin can facilitate the learning of OCL by the students. The selective visualization allows a greater concentration and effectiveness of the analysis of the pathway performed by the queries.	The students in my class seemed to assimilate well the connection between the path highlighted in the diagram and the path made by the OCL queries.

from more recent school years. Finally, it is also planned an experimental validation to measure the difference between the physiological effort, using eye-tracking technology (pupils' fixations and saccades), when developing expressions with and without the plugin.

VI. ACKNOWLEDGMENT

This section is blank due to MODEL's double-blind review process.

REFERENCES

[1] A. Zamansky, G. Rodriguez-Navas, M. Adams, and M. Spichkova, "Formal methods in collaborative projects," in *ENASE*, 2016, pp. 396–402.

[2] M. Gogolla, "Benefits and problems of formal methods," in *Reliable Software Technologies - Ada-Europe 2004*, A. Llamosí and A. Strohmeier, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–15, ISBN: 978-3-540-24841-5.

[3] "Object Constraint Language, version 2.4," Tech. Rep. February, 2014. [Online]. Available: https://www.omg.org/spec/OCL/About-OCL/%20http://www.omg.org/spec/OCL/2.4/PDF.

[4] B. Dobing and J. Parsons, "How uml is used," *Communications of the ACM*, vol. 49, no. 5, pp. 109–113, 2006.

[5] *USE: The UML-based Specification Environment*, Available: https://sourceforge.net/p/useocl/wiki/ Accessed: 2018-12-09. [Online]. Available: http://useocl.sourceforge.net/w/index.php/Main%7B%5C_%7DPage.

[6] A. Nugroho and M. R. Chaudron, "A survey into the rigor of uml use and its perceived impact on quality and productivity," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, ACM, 2008, pp. 90–99.

[7] A. Nugroho, "Level of detail in uml models and its impact on model comprehension: A controlled experiment," *Information and Software Technology*, vol. 51, no. 12, pp. 1670–1685, 2009.

[8] Y.-G. Guéhéneuc, "TAUPE: Towards Understanding program comprehension," in *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research - CASCON '06*, New York, New York, USA: ACM Press, 2006, p. 1.

[9] S. Yusuf, H. Kagdi, and J. I. Maletic, "Assessing the comprehension of UML class diagrams via eye tracking," in *IEEE International Conference on Program Comprehension*, 2007, ISBN: 0769528600.

[10] L. C. Briand, Y. Labiche, H. D. Yan, and M. Di Penta, "A controlled experiment on the impact of the object constraint language in UML-based maintenance," in *IEEE International Conference on Software Maintenance, ICSM*, 2004, pp. 380–389, ISBN: 0962-8452 (Print)\n0962-8452 (Linking).

[11] L. C. Briand, Y. Labiche, M. Di Penta, and H. Yan-Bondoc, "An experimental investigation of formality in UML-based development," *IEEE Transactions on Software Engineering*, 2005, ISSN: 00985589.

[12] K. Siau and P.-P. Loo, "Identifying difficulties in learning uml," *Information Systems Management*, vol. 23, no. 3, pp. 43–51, 2006.

[13] O. Gilles and J. Hugues, "Validating requirements at model-level," *Ingénierie Dirigée par les modèles (IDM'08)*, pp. 35–49, 2008. [Online]. Available: http://www.academia.edu.

[14] T. Yue and S. Ali, "Empirically evaluating ocl and java for specifying constraints on uml models," *Software & Systems Modeling*, vol. 15, no. 3, pp. 757–781, Jul. 2016, ISSN: 1619-1374.

[15] L. Reynoso, M. Genero, and M. Piattini, "Measuring Ocl Expressions: an Approach Based on Cognitive Techniques," in *Metrics for Software Conceptual Models*, Imperial College Press, Distributed by World Scientific Publishing Co., Jan. 2005, pp. 161–206.

[16] L. Reynoso, M. Genero, M. Piattini, and E. Manso, "Assessing the impact of coupling on the understandability and modifiability of OCL expressions within UML/OCL combined models," in *11th IEEE International Software Metrics Symposium (METRICS'05)*, 2005, 10 pp.–14.

[17] L. Reynoso, E. Manso, M. Genero, and M. Piattini, "Assessing the influence of import-coupling on OCL expression maintainability: A cognitive theory-based perspective," *Information Sciences*, vol. 180, no. 20, pp. 3837–3862, 2010, ISSN: 0020-0255.

[18] R. Harrison, S. Counsell, and R. Nithi, "Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems," *Journal of Systems and Software*, vol. 52, no. 2-3, pp. 173–179, 2000.

[19] G. K. Rambally, "The influence of color on program readability and comprehensibility," in *Proceedings of the seventeenth SIGCSE technical symposium on Computer science education - SIGCSE '86*, vol. 18, New York, New York, USA: ACM Press, 1986, pp. 173–181, ISBN: 0897911784.

[20] A. Sarkar, "The Impact of Syntax Colouring on Program Comprehension," *26th Annual Conference of the Psychology of Programming Interest Group*, 2015. [Online]. Available: http://www.ppig.org/sites/ppig.org/files/2015-PPIG-26th-Sarkar.pdf.

[21] R. Schauer and R. K. Keller, "Pattern visualization for software comprehension," in *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98*, IEEE, 1998, pp. 4–12.

[22] C. F. Lange, M. A. Wijns, and M. R. Chaudron, "A visualization framework for task-oriented modeling using uml," in *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, IEEE, 2007, 289a–289a.

[23] R. Mehta and R. Zhu, "Blue or red? Exploring the effect of color on cognitive task performances," *Science*, vol. 323, no. 5918, pp. 1226–1229, Feb. 2009, ISSN: 00368075. arXiv: arXiv:1106.5958.

[24] A. Toval, V. Requena, and J. L. Fernández, "Emerging OCL tools," *Software and Systems Modeling*, vol. 2, no. 4, pp. 248–261, Dec. 2003, ISSN: 1619-1366.

[25] *USE: The UML-based Specification Environment*, Available: https://readable.com/ Accessed: 2019-02-09. [Online]. Available: https://readable.com/.

[26] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Science of Computer Programming*, vol. 69, no. 1-3, pp. 27–34, 2007, ISSN: 01676423.

[27] J. Warmer and A. Kleppe, *The Object Constraint Language Second Edition, Getting Your Models Ready for MDA, by Jos Warmer and Anneke Kleppe.* 6. Addison-Wesley, 2003, vol. 2, p. 139, ISBN: 0321179366.

[28] *JAVA*, Available: https://www.java.com/en/ Accessed: 2019-03-18. [Online]. Available: https://www.java.com/en/.

[29] *Visitor Pattern*, Available: https://sourcemaking.com/design_patterns/visitor Accessed: 2019-03-18. [Online]. Available: https://sourcemaking.com/design_patterns/visitor.

[30] J. Kontio, L. Lehtola, and J. Bragge, "Using the focus group method in software engineering: Obtaining practitioner and user experiences," in *Proceedings - 2004 International Symposium on Empirical Software Engineering, ISESE 2004*, IEEE, 2004, pp. 271–280, ISBN: 0769521657.