Author(s):	Lassila, Matti; Junkkari, Marko; Kekäläinen, Jaana
Title:	Comparison of two XML query languages from the perspective of learners
Year:	2015
Journal Title:	Journal of Information Science
Vol and number:	41 : 5
Pages:	584-595
ISSN:	1741-6485
Discipline:	Computer and information sciences
School /Other Unit:	School of Information Sciences
Item Type:	Journal Article
Language:	en
DOI:	http://dx.doi.org/10.1177/0165551515585259
URN:	URN:NBN:fi:uta-201509252316
Subject:	XML-kyselykielet; opetus; oppiminen; kyselykielten opetus; XML query languages; teaching/learning strategies; pedagogical issues; query languages

Article

JIS

Journal of Information Science
1–13

DOI: 10.1177/0165551510000000
jis.sagepub.com
SAGE

Comparison of two XML query languages from the perspective of learners

Matti Lassila
School of Information Sciences
University of Tampere

Marko Junkkari
School of Information Sciences
University of Tampere

Jaana Kekäläinen
School of Information Sciences
University of Tampere

Abstract
Two XML query languages were tested for intuitivity, learnability and memorability. The languages differ with relation to the query structures like the use of variables, iterators, and reference to attributes. One of the languages, XQuery, is a procedural, expressive and data-oriented query language that is suitable even for programming purposes; the other, XIL, is more declarative, document-oriented query language with a simpler syntax. A query writing test with the learners of the languages was executed. The study indicates that in the query writing, the more procedural query language yields in a greater number of correct queries. Similarity between the tested languages, and to SQL, is discussed from the point-of-view of learnability.

Keywords
Query languages; XML query languages; pedagogical issues; teaching/learning strategies.

1. Introduction

The era of Internet has brought about a need for a common exchange language in order to represent data in a standardized manner and share data between applications. XML has established its status as the de facto standard for this purpose. XML has several advantages, for example being human- and machine-readable it suits different types of Internet applications. The data carried in the XML format varies from strongly to weakly structured data, in other words, from 'database data' (*data-centric*) to documents (*document-centric*), and so vary the use cases. [1,2] Users can be divided into those who manage storing of data and design data structures, and into those who retrieve data. A user may, of course, appear in both roles. In the present study, we focus on learners of XML query languages, who aim to master especially XML structures and retrieval. We analyze which features of the given XML languages the learners adopt easily and which features hamper learning.

Among XML query languages XPath [3] and XQuery [4], provided by WWW consortium, are prevalent. XPath is a path-oriented language with the primary purpose of accessing parts of an XML document. XQuery is a more extensive query language with variables and functions, encompassing XPath as a method for navigating in hierarchical document structures. XQuery is based on earlier XML query languages, like Quilt [5], which in turn have a strong rooting in database query languages, SQL in front [4]. For document-centric XML applications XQuery proved to be too restrictive while it lacks text retrieval features like best match searching and relevance ranking. There are a number of

Corresponding author:
Jaana Kekäläinen, School of Information Sciences, 33014 University of Tampere, Finland
jaana.kekalainen@uta.fi

query languages for XML offering text retrieval features, either extending XQuery or some other XML query approach (e.g. [6,7,8]). XQuery itself was also extended with text retrieval facilities under the XQuery Full Text [9].

XIL, XML Information retrieval Language [10], was designed as a simple query language that supports both data- and document-oriented querying of XML documents, which is essential in many data sets combining tabular data and textual sections. For the ease-of-use, XIL follows the early goals of SEQUEL/SQL: block-structured keyword syntax, variable-free query formulation, linear query expression and non-procedural query formulation. The development of the language involves feedback acquired in the user study.

To summarize, XML query languages are either data-oriented or document-oriented, but in the best case they may address both orientations. The latter is desirable because XML supports combining data and text. Further, query formulation in XML query languages may utilize paths, linear SQL-like querying, variables and iterators (borrowed from logic or programming languages). We explore the intuitivity, learnability and memorability of the specific features by comparing two query languages, XQuery and XIL, in query understanding and query writing tests.

In the next section, we introduce user studies concerned with SEQUEL/SQL and XML query languages because these languages are related to our study. In Section 3, the query languages to be tested are introduced as well as the test setting. In Section 4, we give the results, in Section 5 results are discussed and Section 6 concludes the article.

2. User studies on query languages

The empirical user studies of query languages are rooted in the area of research addressing programming and design of information systems as a psychological phenomenon. Weinberg [11] outlined a generally followed course for the studies. SQL -- and its predecessor SEQUEL -- has been a pair in early comparisons [12,13], later on, query languages based on the syntax of SQL and those designed for XML have been tested [14,15]. The studies apply methods adopted from behavioural sciences to the research of query languages [16]. Typically, the aim has been to evaluate or compare query languages for ease-of-use [13,14,15,17], study query language design and human factors related to it [12,16,17,18,19] or explore human behaviour in query writing/reading [14,19].

The central concepts of these studies deserve consideration. A *query language* refers to a special purpose language, whereby queries may be constructed to retrieve information from a database [20]. It may be understood as a special case of a programming language, with restricted expressive power [16,21]; yet the difference between programming and query languages is not exact (e.g. the case of XQuery). A programming language, in turn, is a language with which one typically gives a procedural description of how to accomplish some operation on a computer.

Procedurality and *declarativity* are features of query and programming languages; roughly, the former means describing a procedure step by step, in other words, how to reach the expected result; the latter means describing the result instead of describing how to compute it. Procedurality and declarativity are often used to describe query languages and their ease-of-use [13]. The issue of declarativity versus procedurality has been debated with no agreement: some researchers are in favour of declarativity because it allows more people to become active in programming [22]; others think that procedural way of thinking is essential for programming [23]; yet others are for a purposeful combination of these features in programming and query languages [24].

Human factors or *human behaviour* in the query language context refer to the behaviour of the users in learning, understanding and using the language. Behaviour is observed through users' actions. *Ease-of-use* is a somewhat illusive concept, which is related to human factors. Ease-of-use is mostly operationalized through comparison: how fast the subjects are able to write queries in each language, how many errors they make per query, how well they can interpret queries in different languages, what is the user experience of the language. Declarative languages are often thought to be easier to use than procedural languages. *Users* are often described by their experience or knowledge with respect to the utilization of the given language. The basic grouping is into experts and novices. This is a rather coarse division and it is further refined by, for example, the frequency of the use of the query language, the type of duties related to querying, subject knowledge (as opposed to IT knowledge). An example of a user grouping comes from Elmasri and Navathe [25]: casual end users who access the database occasionally; naïve or parametric end users who access the database regularly with standard queries; sophisticated users who implement their own applications; stand-alone users maintain personal databases with ready-made programme packages.

The test settings of the user studies on SQL/XML query languages have several features in common. Subjects are often recruited among university students or staff [14,15,19], but in earlier studies they have not always been from computer or information science departments [12,13]. Subjects are either learners of the query language or they have been taught the language for the test purposes. In some studies, users are grouped by their programming experience [12,13] or other knowledge related to the querying or databases [19][1].

The tasks of the tests include writing queries in a query language on the basis of a natural language statement [12,13, 14,15,19] and often also interpreting queries written in a query language into natural language [12,19]. In all studies, test tasks are graded according to their difficulty. The correctness of the answers is judged either on a binary scale (right-wrong, reading tasks [12,19], or gradual scale (e.g. correct, a minor result error, a minor syntax error, a syntactically correct query returns a wrong result, a major [syntax] error by Reisner and others [12]; similar scales utilized by Graaumans [14]; Sengupta and Ramesh [15]).

To summarize the results of the introduced studies [12;13;14;15;16;19] we can state that in the case of SQL or SQL-like query languages, the subjects with no programming skills answered correctly for 44.4-65% of the test tasks; subjects with programming skills succeeded in 54.7-78% of the test tasks.[2] In case non-SQL-like query languages, the subjects with little or no programming skills had correct answers in 47.5-79% of the tasks, and subjects with programming skills succeeded in 57-92% of the tasks. There is a slight tendency for the favour non-SQL languages, which may be described more procedural than SQL (e.g. XQuery, XSLT, [14]).

3. Testing XQuery and XIL

Our study aims to explore whether XIL is easier for learners to adopt than XQuery, and possibly get feedback for further development. Although SQL-based query languages have been studied in user tests (e.g. [14]), the features adopted from SQL differ in the languages. Thus, the earlier tests give guidelines but are not directly comparable.

In the next section, we introduce the query languages and the features to be explored. In Section 3.2, we introduce the test setting.

3.1. XML query languages XQuery/XPath and XIL

XQuery is designed to be a general query language for XML documents, capable for purposes of retrieving and compiling information from heterogeneous sources [26]. Despite of the aim for generality, the development of XQuery has been guided by database use scenarios [27]. Later on, it has been supplemented by a full text (FT) version offering functionality needed for text retrieval (e.g. keyword search, proximity operators, an option for relevance ranking). XQuery is a powerful, Turing complete language that can be used as a programming language [28,29]. Because of the great capability of XQuery and because it has been developed in a fairly diverse team with somewhat competing goals, the language is rather complex [26,27]. XQuery includes XPath, a query language developed for accessing XML hierarchy by path expressions, which may be serialized. The syntax of XQuery follows FOR-LET-WHERE-ORDER-RETURN structure (FLWOR), by which the XML hierarchy is explicitly specified. Since XPath queries are XQuery queries as well, we refer to both as XQuery/XPath.

XIL is an XML query language proposal with the aim to combine a simple text query language with data querying functionality. XIL is not a programming language and its data querying functionality is intentionally restricted for ease-of-use. The syntax of XIL is adopted from SQL: a query is compiled of SELECT-FROM-WHERE blocks, in terms of which queries can be written without the explicit specification of the XML hierarchy. The syntax of XIL allows only linear path expressions. In addition, XIL does not have variables and it is more declarative than procedural. (For sample queries in XIL and XQuery/XPath, see Appendix A.)

3.2. Test setting

The aim of testing is to find out how effective and efficient the query languages are. The use scenario is the one of the learners of the language. As natural foreign languages, query languages are easier to read (interpret) than write (produce). The task of query writing is a realistic task and most often utilized in user tests; the task of reading queries is realistic in a sense that the correct interpretation must precede the correct production of queries. The present study focuses on query writing as it is a more comprehensive test. Effectiveness is operationalized as the correctness of the answers, efficiency as the time spent.

The test participants were taught both languages in a course entitled *XML information retrieval and query languages*. This is a Master's level course and the test was an obligatory part of the course. The query language instruction consisted of lessons and weekly exercises. The course material and exercises were in a web learning environment[3]. Presence in lessons was not an obligation but some of the exercises ware obligatory for passing the course. There were three lessons on XQuery/XPath and one on XIL. All exercises were about the themes of the lesson of the week. The obligatory exercises were about the query languages, XQuery/XPath and XIL, and they were query-writing tasks. The

students were asked to write queries in the given query language corresponding to given statements in natural language. (See Figure 1A for XPath and Figure 1B for XIL. NB: The tasks in the figures are not the same.) For XQuery/XPath and XIL, the writing tasks were 'cloze tests' where a part of the query was given and the missing part had to be replaced (see Figure 1B). The correctness of the queries was checked automatically, and in case of an incorrect query the students were allowed to re-enter it, guided by error messages. There were 20 exercises for each language; the natural language statements for XIL and XQuery/XPath were identical.

Figure 1. XPath (A) and XIL (B) exercises in the learning environment

The test was implemented as a 'paper and pen' exercise because we did not have a XIL application supporting several users simultaneously. The test tasks were on a paper sheet, 15 tasks per language. The tasks were natural language statements that were to be translated into a given query language. The statements were identical in XQuery/XPath and XIL (see Appendix A). The queries were targeted to the test material, consisting of two XML documents: one was an excerpt of a play, the other included dates and headings of some cables relating to diplomacy.

We considered the *complexity of the test tasks* and the *difficulty of the model queries* because these are likely to affect the results. Methods to calculate the complexity of queries have been suggested, for example, by Orman [30], Chan [31], and Graaumans [14]. A *complexity* figure was calculated for the test tasks according to the Graaumans's method [14]. The complexity is the sum of elements, attributes and the values that should be returned as a result, plus the number of conditions that were given in the task. Complexity is a quantity independent of the query language or documents. [32] The tasks were arranged from the least complex to the most complex on the test sheet. *Difficulty* is adopted from Halstead [33] who originally used this concept for a programme code. The calculation of difficulty is explained in Appendix B, and the complexity and difficulty of the tasks is given in Table 6, Appendix C.

Altogether 39 subjects[4] participated in four tests, which were all identical in their arrangements. The goal of the test was explained to the subjects and they were given the test material. All subjects were supposed to write 15 queries in XIL and XQuery/XPath. Because they were free to leave at their own discretion, all tasks were given to each subject immediately. For 20 students, the test material was arranged such that XIL queries appeared first, for 19 students XQuery/XPath queries were first. Nevertheless, the subjects were free to write the queries in any order. The time when the subject received the task form and the time when he returned it was recorded. Unfortunately, the time measurement is not language bound because of the test arrangement. The forms were returned with the names of the subjects because

Table 1. Correct, erroneus and empty queries by query languages.

Task #	Correct queries		Erroneous queries		Empty queries	
	XIL	XQuery/XPath	XIL	XQuery/XPath	XIL	XQuery/XPath
1	30	34	5	5	4	0
2	12	30	23	8	4	1
3	3	15	32	23	4	1
4	3	13	31	24	5	2
5	6	12	28	25	5	2
6	13	21	21	16	5	2
7	11	4	22	33	6	2
8	8	1	24	32	7	6
9	1	14	31	20	7	5
10	25	6	7	28	7	5
11	4	16	28	20	7	3
12	1	8	28	22	10	9
13	8	0	23	33	8	6
14	7	4	23	26	9	9
15	1	4	30	27	8	8
Total	133	182	356	342	96	61

we wanted to use other course information as variables in the test. Yet, the subjects had an option to forbid the use of their answers and information in the test. All subjects agreed to participate.

4. Results

The subjects of the test were a fairly homogenous group by their background: 29 subjects (74%) had computer science as the major subjects; over 30 subjects (87-94%) had participated in the preceding bachelor level courses; 28 subjects (72%) programmed professionally or as a hobby.

4.1. Test performance and background variables

The number of correct, erroneous and empty queries by the query languages is given in Table 1.The performance varied significantly by the query languages, for the benefit of XQuery/XPath (Wilcoxon signed rank test, $p<0.02$ for correct queries and $p<0.009$ for erroneous queries).

To test the possible effects of the subjects background, we considered subjects' activity in programming outside studies (voluntary programming), their familiarity with SQL (binary variables), time spent doing web exercises during the course, time spent in the test (minutes, continuous varibale) and the course grade (ordinal variable) which we considered to be a proxy for students' commitment and motivation. The order of the test tasks (XIL or XQuery/XPath first) was also taken into account. The number of correct queries in each language was the dependent variable in a multiple regression test. The regression results are given in Table 2. The model accounts for 48% (XIL) and 45% (XQuery/Xpath) of the variation in performance. Especially course grade and voluntary activity in programming are good predictors.

Table 2. Multiple regression analysis of background variables and correct queries (N=33, six cases deleted due missing values).

R-squared	XIL 0.48 (p<0.01)				XQuery/XPath 0.45 (p<0,05)			
	Est.	Std.err.	t	p	Est.	Std.err.	t	p
Volunt. prog.	2.53	1.09	2.33	0.03*	3.49	1.35	2.59	0.02*
Fam. SQL	-2.08	1.09	-1.92	0.07	-1.22	1.35	-0.91	0.37
Study time	0.02	0.01	1.29	0.21	-0.02	0.01	-1.27	0.22
Test time	0.04	0.03	1.16	0.26	0.04	0.04	1.01	0.32
Task order (XQ/XP)	0.35	0.88	0.40	0.69	1.11	1.09	1.02	0.32
Grade	1.47	0.40	3.69	0.00**	1.14	0.49	2.32	0.03**

*p<0.05; **p<0.01

Table 3. Error types in XIL and XQuery queries, their total number (Σ), and the number of subjects who committed them (n).

Error type: XIL	n	Σ	Error type: XQuery/XPath	n	Σ
Path error	33	113	Misc. syntax error	59	187
Attribute error	26	80	Path error	31	94
Language loan	25	118	Result formulation error	17	40
Misc. syntax error	23	37	Predicate clause error	14	36
Missing data	13	24	Semantic error	11	18
Reference error	11	22	Comparison error	9	12
Semantic error	9	11	Inadequate solution	7	8
Language confusion	3	29	Language loan	6	16
Missing part	3	3			

4.2. Error types by languages

All the queries (N=585/query language) were analysed; first by running the queries in a query language interpreter, then by analysing the error types in the queries. A data-driven classification was constructed for the error types (Table 3). Each erroneous query could fall into several classes because one query might include several errors types. The classes differ between the languages because of the different structure and style of the languages. This analysis gives an overview of the errors committed in each language. Some errors are frequent but committed by few subjects, some errors are common and committed by most subjects; thus, besides the total number of errors, the number of subjects committing the error is given in Table 3.

In XIL queries the most frequent error type was a path error (Table 3). In this error type, the path given in the query does not refer to any element in the document. This error seems to be caused by confusion: partly because SQL syntax has no paths, partly because the syntax of XPath differs slightly from the syntax of XIL. Attribute errors and language loans are typical errors committed by several subjects. Attribute errors can be divided into three main sub-classes: an attribute without the including element, an attribute addressed as an element, and an attribute without a separator. These again may be caused by the differences in addressing attributes between XIL and XQuery/XPath: in the former, addressing an attribute in a query path returns the elements encompassing the attribute (e.g. `//paragraph/@style="line"`); in the latter, the same query returns a truth value. In XQuery/XPath, a query resulting the same output as in XIL would be `//paragraph[@style="line"]`, i.e. it requires a serialized path expression. Sometimes, the FROM-WHERE structure of SQL was applied to XIL when it was not needed.

A language loan occurs when an expression, operator or some other feature of another language is used when writing a query in a given language. Typical loans are predicate clauses and 'contains' function from XQuery/XPath. A language confusion error is committed when the whole query is written in another language than expected.

Miscellaneous syntax errors in XIL are an aggregated class for several syntax mistakes. Missing data refers to cases where a query does not include all elements or attributes required in the query specification. A reference error occurs typically when the FROM part of a query refers to an element too high in the document hierarchy. The query returns all the siblings instead of one element. A semantic error is committed when the result does not correspond to the task given although the syntax of the query could be correct. A missing part refers to cases where some part of the query, like FROM/WHERE part, is missing. (For sample errors, see Appendix D.)

The most typical errors in the XQuery/XPath queries were different syntax errors, which is understandable because of the more complicated structure of XQuery/XPath. The frequency of path errors is about the same as in XIL queries although the reasons are different: the path may be syntactically correct but the referent does not exist. Error in formulating a result is typically in RETURN part where the subject has forgotten to add brackets. The predicate error occurs when the query returns a truth value when the goal was to return an element with the predicate condition. Semantic errors and language loans are similar to those in XIL though the latter errors are more common in XIL. In a comparison error an expression referring to several elements in an XML hierarchy is compared with a value, which is not allowed in XQuery/ XPath. An inadequate solution does not give all information required in the task.

Table 4. Query error classification according to Welty and Stemple (1981).

Error type	Description
Correct	The query is correct.
Minor syntax error	The query has a minor syntax error that can be automatically corrected.
Minor operand error	The query has a minor error in the form of the operand, e.g. spelling error in the name of the operand.
Minor semantic error	The query is syntactically correct but does not return the expected result because the subject has misunderstood the task.
Correctable	The query has an error that is correctable with a feedback given by the system.
Semantic error	The query is syntactically correct but does not return a correct result.
Syntax error	The query has a syntax error and is not executable.
Inadequate solution	The query does not fulfil all the requirements given in the task description.
Empty query	The subject has not formulated a query.

4.3. Comparison of errors between the languages

The classifications of the languages in Table 3 are not commensurable. In order to compare the languages, the errors were classified according to query error categories introduced by Welty and Stemple ([13], Table 4). This classification is exclusive and independent of query languages. The queries in the first four classes are regarded as practically correct. A crucial difference is between correctable (from the second to the fifth class) and non-correctable (the last four classes) queries; the former could be saved, for example by giving automatic feedback.

Table 5 shows that the number of correct and correctable queries is larger among XQuery/XPath than XIL queries, and the number of negative cases is larger among XIL queries. Also here, the error types differ between languages: semantic errors are few in XQuery/XPath whereas in XIL they are common; syntax errors are more frequent in XQuery/XPath than in XIL queries. In some test tasks the correctness of the queries varied hugely by language: for the test *tasks 10* and *13*, the numbers of correct XIL queries were 25 and eight against six and zero correct queries in XQuery/XPath; for the test *tasks 9* and *11*, the number of correct queries were 14 and 16 in XQuery/Xpath, but only one and four in XIL. (Table 1 and Appendix A.)

4.4. Halstead difficulty and test performance

The relation between the difficulty of the tasks and test performance was established with regression analysis. The working hypothesis is that the more difficult the model query in a given language, the less the subjects return correct queries. The regression model is simple linear regression with the Halstead difficulty as the independent variable; the dependent variable is the number of the returned correct queries of the given difficulty figure. Pearson correlations and regression lines are shown in Figure 2.

Table 5. Correct and erroneous test queries according to Welty & Stemple (1981) classification.

Error type	XIL	XQuery
Correct	133	182
Minor syntax error	0	12
Minor operand error	0	4
Minor semantic error	0	1
Correctable	112	76
Semantic error	110	37
Syntax error	133	205
Inadequate solution	1	7
Empty query	96	61
Total	585	585

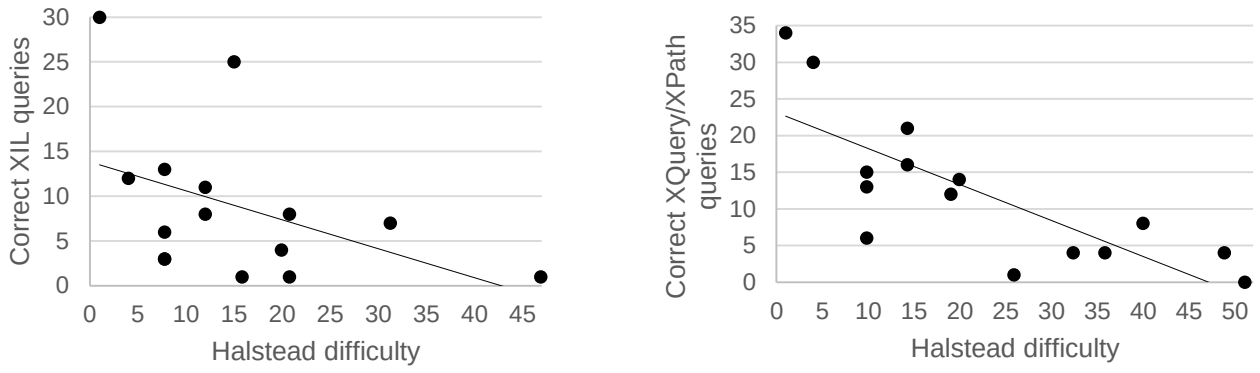

Figure 2. Regression analysis of Halstead difficulty and test performance.

Obviously, Halstead difficulty predicts the performance in XQuery queries better (R-squared 0.6, p<0,001) than in XIL queries (R-squared 0.2, no significance). It seems that difficulty of the queries always explains the differences in performance to a certain degree but in case of XIL other aspects affect the performance stronger. In the next section, we discuss the possible explanations.

5. Discussion

The XIL query language was designed to combine document-oriented information retrieval features with data-oriented querying with a simple syntax. According to the test results, XIL was not easier for learners because it deviated from their earlier experience with query languages and thus from their expectations: similarity to SQL and XPath did not work for XIL, rather it led the learners astray. The learners expected exact sameness and did not memorize the features that were meant to facilitate query writing. This is a type of semantic overloading: same structures are used for slightly different purposes.

Most common error types, attribute and path errors, the participants committed in XIL queries are also the most prevalent correctable errors. These errors are clustered in tasks 3, 4, 9, 11 (see Appendix A and C). It is worth noting that the model queries of these tasks are not the most difficult in Halstead rating. Obviously, the Halstead difficulty predicts performance when the users master the language, but in the XIL case certain features not mastered by the subjects affected the performance more. It is possible that the established query languages, like XQuery, are in favour with the computer science students, who also have more experience of those languages, which would partly explain the better performance in XQuery.

The declarative features of XIL outperformed the procedurality of XQuery in the test tasks 10 and 13, but in general the learners had assimilated procedural thinking and did not benefit from declarativity. For both query languages, the voluntary activity in programming and the course grade were good predictors for the performance in writing correct queries. These indicators are connected to learners' motivation and commitment and are rather intuitive causes.

It is debatable whether 'paper and pen' query writing tasks are realistic since there is no interaction or reference material, unlike in real situations. Nevertheless, this type of task reveals learning styles, memory issues and mindsets. It also demonstrates under simplified conditions what is easy to learn.

6. Conclusions

We conducted a test to compare two query languages with relation to the query writing capability of the learners of the languages. Our aim was to explore which features make a query language easy to learn and use. We compared XIL, a new, declarative, SQL-like XML query language, and XQuery, an established, procedural XML query language, with a more complex, SQL-inherited syntax. The declarative features and the expectedly simpler syntax of XIL did not yield in better performance. In query writing, XQuery turned out to yield more correct queries. It seems that the earlier experience of query and programming languages strongly affects the performance in query writing. The difficulty of the queries predicts the performance in query writing in case XQuery; in the case of XIL the similarity of the syntax to

XPath and SQL with some crucial differences led the learners astray and caused more errors. The results are in line with earlier user studies on query languages.

A pedagogical conclusion is that semantic overloading of any features of the query languages is to be avoided. In other words, similar primitives should not be used in different intentions. This has implications for the development of the language as well: 1. In the observed cases of the semantic overload the syntax of commands must be modified. 2. The functionality of basic operations should correspond to generally accepted notational conventions.

The test setting was biased to abilities and thinking necessary for the students of computer science. We believe that the era of big and open data will bring about new users, like data journalists, for query languages. We aim to develop XIL further to meet the needs of the new user groups. As a further study, we aim to test a new version of XIL with subject professionals with less IT knowledge.

Notes

1. It is worth noting that the criteria for expertise or experience have varied over time, e.g. in the early studies the programmer status could be gained by attending one programming course.
2. Graaumans ([14], SQL/XML) reported results for easy and difficult tasks separately. Correct answers were given in 94% of easy tasks and 42% of the difficult tasks.
3. Moodle (https://moodle.org)
4. Most of the subjects were males. We did not test the differences between the genders.

References

[1] DeRose S. Navigation, access and control using structured information. American Archivist 1997; 60: 298–309.
[2] Salminen A and Tompa F. Communicating with XML. Heidelberg: Springer, 2011.
[3] Berglund A, Boag S, Chamberlin DD, et al. 'XML path language (XPath) 2.0 (Second Edition)'. W3C Recommendation. http://www.w3.org/TR/xpath20/ (2010, accessed October 2014).
[4] Boag S, Chamberlin D, Fernández M, Florescu D, Robie J and Siméon J. 'XQuery 1.0: An XML query language (Second Edition)'. W3C Recommendation. http://www.w3.org/TR/xquery/ (2010, accessed October 2014).
[5] Chamberlin D, Robie J and Florescu D. Quilt: An XML query language for heterogeneous data sources. In: Goos G, Hartmanis J, van Leeuwen J, Suciu, D and Vossen G (eds) The World Wide Web and Databases. Vol. 1997. Heidelberg: Springer, 2001, pp. 1-25.
[6] Carmel D, Maarek YS, Mandelbrod M, Mass Y and Soffer A. Searching XML documents via XML fragments. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval (SIGIR '03). New York USA: ACM Press, 2003, pp. 151-158.
[7] Fuhr N and Großjohann K. XIRQL: An XML Query language based on information retrieval concepts. ACM Transactions on Information Systems 2004; 22: 313–356.
[8] Bremer J-M and Gertz M. Integrating document and data retrieval based on XML. The VLDB Journal 2006; 15: 53–83.
[9] Case P, Dyck M, Holstege M, Amer-Yahia S, Botev C, Buxton S, et al. (eds) 'XQuery and XPath Full Text 1.0'. W3C Recommendation 17 March 2011. http://www.w3.org/TR/xpath-full-text-10/ (2011, accessed October 2014).
[10] Junkkari M, Arvola P and Kekäläinen J. Grammatical approach to XML information retrieval query languages. Technical report A-2006-5. Department of Computer Sciences, University of Tampere, 2006.
[11] Weinberg GM. The psychology of computer programming. New York, NY: Van Nostrand Reinhold, 1971.
[12] Reisner P, Boyce R and Chamberlin D. Human factors evaluation of two data base query languages: Square and Sequel. In: AFIPS12 '75: Proceedings of the national computer conference and exposition. New York: ACM, 1975, pp. 447–452.
[13] Welty C and Stemple D. Human factors comparison of a procedural and a nonprocedural query language. ACM Transactions on Database Systems 1981; 6: 626–649.
[14] Graaumans J. 'Usability of XML query languages.' SIKS Dissertation Series 2005-16. http://igitur-archive.library.uu.nl/dissertations/2005-1018-200002/full.pdf (2005, accessed October 2014).
[15] Sengupta A and Ramesh V. Designing document SQL (DSQL): An Accessible yet comprehensive ad-hoc querying frontend for Query. Journal of Database Management 2009; 20: 26–53.
[16] Reisner P. Human factors studies of database query languages: A survey and assessment. ACM Computing Surveys 1981; 13: 13–31.
[17] Chan H, Siau K and Wei K-K. The effect of data model, system and task characteristics on user query performance – An empirical study. The Data Base for Advances in Information Systems 1998; 29: 31-49.
[18] Yen MY-M & Scamell RW. A human factors experimental comparison of SQL and QBE. IEEE Transactions on Software Engineering 1993; 19: 390–409.
[19] Weiand K. 'Keyword-based querying for the social semantic web.' Ludwig-Maximilians-Universität, München. http://edoc.ub.uni-muenchen.de/12671/ (2010, accessed October 2014).

[20] Reisner P. Query Languages. In: Helander M. (ed) Handbook of human-computer interaction. Amsterdam: Elsevier, 1988, pp. 257–280.
[21] Nardi B. A small matter of programming: perspectives on end user computing. Cambridge, MA: MIT Press, 1993.
[22] Lloyd JW. Practical advantages of declarative programming. In: Alpuente M., Barbuti R., Ramos I. (eds) 1994 Joint Conference on Declarative Programming, GULP-PRODE'94. Volume 1. Peñiscola, Spain, 1994, pp. 18-30.
[23] Soloway E. Kill two birds with one stone: Teach procedural query languages. In: Proceedings of the ACM ’82 conference (ACM '82). New York, NY: ACM, 1982.
[24] Manthey R. Declarative languages - paradigm of the past or challenge of the future? In: Proceedings of the 1st International East-West Database Workshop. LNCS 504. New York, NY: Springer Verlag, 1991, pp. 1–16.
[25] Elmasri R and Navathe S. Fundamentals of database systems. 6th ed. Boston: Addison-Wesley, 2011.
[26] Chamberlin D. Influences on the design of XQuery. In: Katz H. (ed) XQuery from the experts: A guide to the W3C XML query language. Boston: Addison-Wesley, 2003, pp. 81-141.
[27] Kay M. XQuery, XPath and XSLT. In: Katz H. (ed) XQuery from the Experts: A Guide to the W3C XML query language. Boston: Addison-Wesley, 2003, pp. 145–183.
[28] Kepser S. A Simple Proof for the Turing-completeness of XSLT and XQuery. In: Proceedings of Extreme Markup Languages, 2004.
[29] Bamford R, Borkar V, Brantner M, Fischer P, Florescu D, Graf D, et al. XQuery reloaded. In: Proceedings of the VLDB Endowment 2, 2009, pp. 1342–1353.
[30] Chan HC. The relationship between user query accuracy and lines of code. International Journal of Human-Computer Studies 1999; 51: 851–864.
[31] Orman L. Complexity of database languages. Information Systems 1991; 16: 169–184.
[32] Graaumans J. ‘A collection of XML documents and query tasks.’ UU-CS-2005-038. Utrecht University: Institute of information and computing sciences. http://igitur-archive.library.uu.nl/math/2007-1127-200943/UUindex.html (2005, accessed October 2013).
[33] Halstead MH. Elements of software science. New York, NY: Elsevier, 1977.

Appendices

A. Test tasks and model queries

Test tasks with model queries in XIL and XQuery/ XPath (alternatives). Note that there are several ways to write a correct query, model queries are not the only solutions. ‘Play’, ‘cable’ etc. in tasks refer to the test material: one text sheet containing cables and one text sheet containing an excerpt of a play. For clarity, the reserved query words are in capital letters in model queries.

1. Write a query that returns all paragraphs from the play.
XIL: SELECT paragraph
XPath: play//paragraph
XQuery: FOR $p in play//paragraph RETURN $p

2. Write a query that returns section titles.
XIL: section/title
XPath: //section/title
XQuery: FOR $t in //section/title return $t

3. Write a query returning all paragraphs the style of which is *line*.
XIL: SELECT paragraph/@style=line
XPath: //paragraph[@style="line"]
XQuery: FOR $p in //paragraph WHERE $p/@style="line" RETURN $p

4. Write a query that returns all cables classified *secret*.
XIL: SELECT cable/@classification=secret
XPath: //cable[@classification="secret"]
XQuery: FOR $b IN //cable WHERE $b/@classification="secret" RETURN $b

5. Write a query that returns the title of a section containing the word *song*.
XIL: SELECT title FROM section ABOUT song
XPath: //section[CONTAINS(.,"song")]/title **XQuery**: FOR $s IN //section[CONTAINS (.,"song")] RETURN $s/title

6. Write a query that returns a subject of a cable, the subject containing the word *war*.
XIL: SELECT subject ABOUT war FROM cable
XPath: //cable/subject[contains(.,"war")]

7. Write a query that returns the subjects of the cables sent or published in 2009.
XIL: SELECT subject FROM cable WHERE year=2009
XPath: //cable[published/year="2009" OR sent/year="2009"]/subject

8. Write a query that returns titles and keywords for acts containing the word *missus*.
XIL: SELECT title, keyword FROM act ABOUT missus
XPath: //act[CONTAINS(.,"missus")]/(title|keyword)
XQuery: FOR $a IN //act WHERE CONTAINS ($a, "missus") RETURN {$a/title} {$a/keyword}

9. Write a query that returns the names of the recipients of confidential cables.
XIL: SELECT to/name FROM cable/@classification=confidential
XPath: //cable[@classification="confidential"]//to/name
XQuery: FOR $a in //cable WHERE $a/@classification="confidential" RETURN $a//to/name

10. Write a query that returns the titles and the descriptors of the acts.
XIL: SELECT act/title, act/descriptor FROM play
XPath: //act/(title|descriptor) / XQuery: FOR $a in //act RETURN ($a/title, $a/descriptor)

11. Write a query returning the introduction of the section, the short title of which is *School*.
XIL: SELECT section/introduction FROM chapter WHERE section/@short_title =School
XPath: //section[@short_title="School"]/introduction

12. Write a query that returns the distribution information of such cables that are sent before 2009. Group the results by cables.
XIL: SELECT distribution FROM cable GROUP BY cable WHERE sent/year < 2009
XQuery: FOR $c IN //cable WHERE $c/sent/year<2009 RETURN <cable>{$c/distribution}</cable>

13. Write a query that returns the title of a section containing morning in the introduction and proposal in any act.
XIL: SELECT title FROM section WHERE introduction ABOUT morning AND act ABOUT proposal
XQuery: FOR $t in //section WHERE CONTAINS($t/introduction, "morning") AND CONTAINS($t/act, "proposal") RETURN $t/title

14. Write a query that returns the subjects of the cables sent before 2009 and with a recipient element containing *WASHDC*.
XIL: SELECT subject FROM cable WHERE sent/year=2009 AND to/name ABOUT WASHDC
XPath: //cable[sent/year < 2009 AND CONTAINS (.//to, "WASHDC")]/subject
XQuery: FOR $c IN //cable WHERE $c/sent/year < 2009 AND CONTAINS(.//to, "WASHDC") RETURN $c/subject

15.Write a query that returns title and keywords for acts with the identifier *act1* or *act2*.
XIL: SELECT act/title, act/keyword GROUP BY act FROM play WHERE act/@indentifier=act1 OR act/@identifier=act2
XPath: //act[@identifier="act1" OR @identifier="act2"/(title|keyword)
XQuery: FOR $a IN //act WHERE $a/@identifier="act1" OR $a/@identifier="act2" RETURN {$a/title, $a/keyword}

B. Applied Halstead difficulty

Let us consider the following query:
SELECT subject **FROM** cable **WHERE** year=2009
The query is interpreted as a sequence of strings which are classified as either operators or operands. The strings **SELECT, FROM, WHERE, =** are operators, and strings **subject, cable, year, 2009** are operands. Now, the number of unique operators (n_1) and operands (n_2) are calculated, as well as the total number of the occurrences of these (N_1, N_2). The length of a query is

$$N = N_1+N_2 \quad (1)$$

The size of the vocabulary is the sum of the unique operators and operands

$$n = n_1+ n_2 \quad (2)$$

The volume of a query is calculated as

$$V = N*\log_2(n) \quad (3)$$

The potential volume of a query is the minimum size for the query

$$V' = n'*\log_2(n') \quad (4)$$

where n' is the size of the minimum vocabulary needed for a query. In case of XQuery and XIL, the size of the minimum vocabulary is two because the simplest query in both languages contains one operand and one operator. Difficulty is

$$D = V/V' = (N*\log_2(n))/(n'*\log_2(n')) \quad (5)$$

For the sample query

$$D = (8*\log_2(8))/(2*\log_2(2)) = 12 \quad (6)$$

C. Complexity and difficulty of the test tasks

Table 6. Complexity (C) and difficulty (D) of the test tasks.

Task #	C	D_{XIL}	D_{XQuery}
1	1	1.0	1.0
2	2	4.0	4.0
3	3	7.8	9.8
4	3	7.8	9.8
5	3	7.8	19.0
6	3	7.8	14.3
7	4	12.0	35.8
8	4	12.0	25.9
9	5	15.8	19.9
10	5	15.0	9.8
11	6	19.9	14.3
12	6	20.8	40.0
13	6	20.8	51.1
14	8	31.3	48.9
15	12	46.9	32.4

D. Sample errors in queries

Table 7. XIL queries.

Task #, error type	Model query	Erroneous query
11, path error & attribute error	SELECT section/introduction FROM chapter WHERE section/@short_title ABOUT School	SELECT introduction FROM section WHERE section/short_title ABOUT "School"
9, attribute error	SELECT recipient/name FROM cable/@classification=confidential	SELECT cable//recipient/name WHERE classification="confidential"
4, language loan	SELECT cable/@classification=secret	select //cable[@classification="secret"]
3, reference error	SELECT paragraph/@style=line	SELECT paragraph FROM book WHERE paragraph/@style="line"
6, syntax error & language loan	SELECT subject ABOUT war FROM cable	SELECT title WHERE exists="war"

Table 8. XPath/XQuery queries.

Task #, error type	Model query	Erroneous query
3, syntax error	//paragraph[@style="line"]	paragraph/[@style="line"]
7, path error	//cable[.//year="2009"]/title	//cable[/released/year=2009]/title
15, result formulation error	//act[@id="act1" or @id="act2"] /(title|keyword)	for $n in act return $n/title, $n/keyword
4, predicate clause error	//cable[@classification = "secret"]	//cable/@classification="secret"
7, language loan	for $cable in //cable[sent/year/text()<2009] return {$cable/distribution}	//cable [sent/year< 2009]/distribution order by cable