XChange: A Semantic Diff Approach for XML Documents

Alessandreia Oliveira[1,2], Troy Kohwalter[1*], Marcos Kalinowski[3], Leonardo Murta[1], Vanessa Braganholo[1]

[1]*Instituto de Computação, Universidade Federal Fluminense, Niterói – RJ, Brazil*
[2]*Departamento de Ciência da Computação, Universidade Federal de Juiz de Fora (UFJF), Juiz de Fora, MG, Brazil*
[3]*Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, RJ, Brazil*

Abstract. XML documents are extensively used in several applications and evolve over time. Identifying the semantics of these changes becomes a fundamental process to understand their evolution. Existing approaches related to understanding changes (*diff*) in XML documents focus only on syntactic changes. These approaches compare XML documents based on their structure, without considering the associated semantics. However, for large XML documents, which have undergone many changes from a version to the next, a large number of syntactic changes in the document may correspond to fewer semantic changes, which are then easier to analyze and understand. For instance, increasing the annual salary and the gross pay, and changing the job title of an employee (three syntactic changes) may mean that this employee was promoted (one semantic change). In this paper, we explore this idea and present the XChange approach. XChange considers the semantics of the changes to calculate the diff of different versions of XML documents. For such, our approach analyzes the granular syntactic changes in XML attributes and elements using inference rules to combine them into semantic changes. Thus, differently from existing approaches, XChange proposes the use of syntactic changes in versions of an XML document to infer the real reason for the change and support the process of semantic diff. Results of an experimental study indicate that XChange can provide higher effectiveness and efficiency when used to understand changes between versions of XML documents when compared with the (syntactic) state-of-the-art approaches.

Keywords: Semantic diff; Match; Similarity; Evolution of XML documents.

* Corresponding Author

1. Introduction

Several systems adopt XML (*eXtensible Markup Language*) [1] to represent semi-structured data. Many industries and the scientific communities have adopted XML documents as a standard for representing, storing, and exchanging data. Consequently, a large amount of XML documents is generated every day. Examples include applications in Web Science [2], in the health care area [3,4], and the legislative branch [5]. Moreover, the Brazilian National Research Council (CNPq) requires every researcher to store their curricula in the Lattes platform [6], which uses XML as an exchange format. Additionally, the *Brazilian Open Data Portal* contains data published by government agencies related to supplementary health, transportation system, security, education, government expenditures, electoral process, etc. [7]. Finally, Wikipedia periodically generates dump files of the data it manages: articles, images, categories, constraints, and other metadata [8]. DBLP does the same for publication data it stores [9,10].

All this data, however, is not static. Similar to data stored in relational databases, semi-structured data also evolves over time. The usage of hierarchical structure and user-defined tags [1] allows for flexibility in data representation. However, this also introduces challenges to compare XML documents, especially in large repositories. This problem has been studied in the literature [11–29]. Unlike traditional textual diff approaches, which consider lines of text files as atomic elements during the comparison of two versions, these approaches are aware of the XML syntax. Thus, they can compare the elements and attributes of an XML document, even if they are all on the same line of the file.

Some of these approaches use similarity calculations [30–34] to compare and identify the corresponding elements across versions, while others use context keys [35,36], which can be expressed in the document's schema [37]. For example, XyDiff [14] uses an ID attribute to match elements across versions of the document. A potential problem, in this case, is that the keys may not be maintained in all situations. Depending on how users manage the XML documents, there is no guarantee that the key values remain the same as the document evolves into new versions. Also, schemas (which is where IDs are defined) are not mandatory, so most XML documents do not use them [38–40].

Even in the cases where there is a unique and stable ID, if a large XML document undergoes many modifications, understanding the underlying meaning of fine-grained syntactic changes is difficult. That is, knowing the reasons behind multiple pseudo-aleatory insertions, removals, and updates of elements and attributes in a large document are not trivial. A key observation that motivated our work is that several granular modifications may together refer to a single modification at the semantic level. For instance, increasing the annual salary and the gross pay, and changing the job title of an employee (three syntactic changes) may mean that this employee was promoted (one semantic change). The existing XML diff-related approaches, including XyDiff [14] and X-Diff [27], which are the two most cited approaches in the literature[2] and the basis for several other proposals, only identify syntactic changes. They do not consider the coarse-grained semantic change that motivated these multiple fine-grained syntactic changes, which varies according to the (knowledge) domain. Given this fact, they are not able to identify the intention behind the changes.

In this work, we propose a semantic diff approach for XML documents, named XChange. The purpose of XChange is to support the understanding of the evolution of two sequential versions (from now on, called *v1* and *v2*) of the same XML document, as far as the semantic diff is concerned. Thus, Xchange aims at identifying the real reason for the modifications that transformed *v1* into *v2*. XChange starts by analyzing the syntax of *v1* and *v2* to find corresponding elements (matches). In this matching phase, the user can choose a matching strategy: *matching by key* or *matching by similarity*. The result of this phase is a set of syntactic differences that XChange incorporates into a knowledge base. XChange then uses a set of rules, previously inserted in the knowledge base, to infer the semantic differences between *v1* and *v2*. The domain expert creates this set of rules, which can vary depending on the domain.

Our approach was designed to work with data-centric XML documents [41], which have a well-defined, regular structure. These documents are usually shallow and wide, and represent data instead of formatting or other syntax-related information.

[2] According to Google Scholar, XyDiff has 614 citations, while X-Diff has 525 (citations collected on August 29th, 2019).

We evaluate XChange through an experimental study where we compare XChange with X-Diff [27] since X-Diff produces more precise results than XyDiff [20]. The goal of this study is to answer the following research questions (RQ): (RQ1) *Is XChange's semantic change identification more effective in providing the means for users to understand the XML document evolution than X-Diff's syntactic changes identification?* (RQ2) *Is XChange's semantic change identification more efficient in providing the means for users to understand the XML document evolution than X-Diff's syntactic changes identification?*

The results of our evaluation indicate that XChange is more effective and more efficient when compared with X-Diff. Even though our proposed approach produces a larger delta file when comparing to X-Diff, the summary and structure of XChange's delta allowed the participants to find the correct answers in a shorter time without having to go through the entire document.

We organize this paper as follows: Section 2 provides a guiding example that we use throughout the paper to explain our approach. Section 3 describes our approach. Section 4 provides an evaluation of our approach and discussion of the obtained results. Section 5 presents the related work. Finally, Section 6 provides conclusions and future work.

2. Motivational Example

Consider an XML document related to employee registration, obtained from the Baltimore City Hall [42]. This XML document has the following data: employee name (<*name*>), job title (<*jobtitle*>), agency code (<*agencyid*>), agency name (<*agency*>), hire date (<*hiredate*>), annual salary (<*annualsalary*>), and gross pay (<*grosspay*>). The XML document is also organized in three depth layers: <*government*>, <*employee*>, and the employee data. Figure 1 illustrates a small but didactic fragment from this XML document with four employees. This fragment represents the first version of the document (*v1*). Another XML fragment of the same document, presented in Figure 2, shows the second version (*v2*) of the XML document, which is a revision of *v1* containing changes in three of the original employees, a removed employee registry, and a new employee registry.

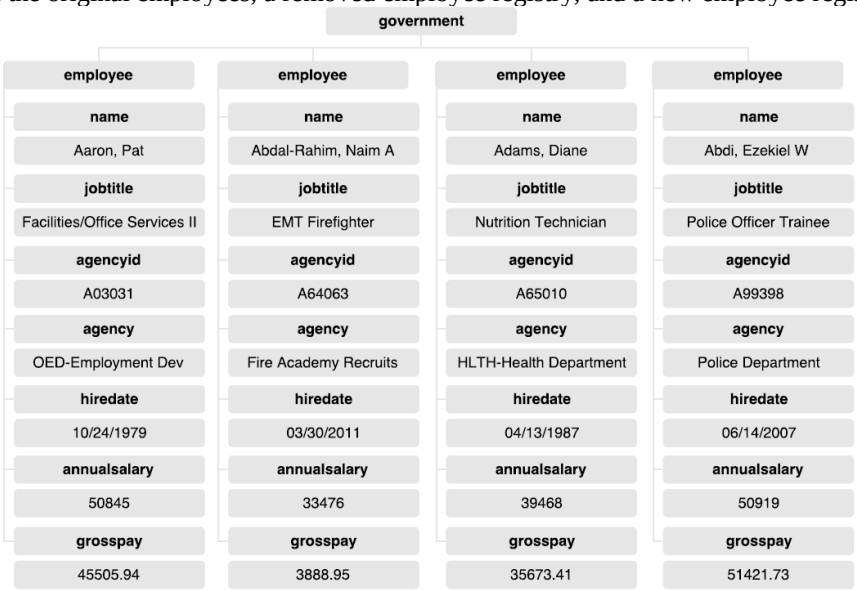


Figure 1: XML Document related to Baltimore's City Hall employee registration at its first version (*v1*)

Figure 2 shows three employees that appear in both versions of the document (*v1* and *v2*) marked in yellow. This yellow marking represents changes made in some elements of the document. For example, employee *Aaron, Pat* had changes in her name to *Aaron, Patricia G*, as well as her gross salary and annual salary. Figure 2 also has green, gray, and red markings. Green markings, as illustrated in the employee named *Aaron, Petra L*, represent new additions that are present only in the current version (*v2*) and not in the previous one (*v1*). Red markings represent deleted elements in the current version (v2), such as employee *Adams, Diane*. Version *v2* is, therefore, a consequence of the evolution of the data of the employees in the document.

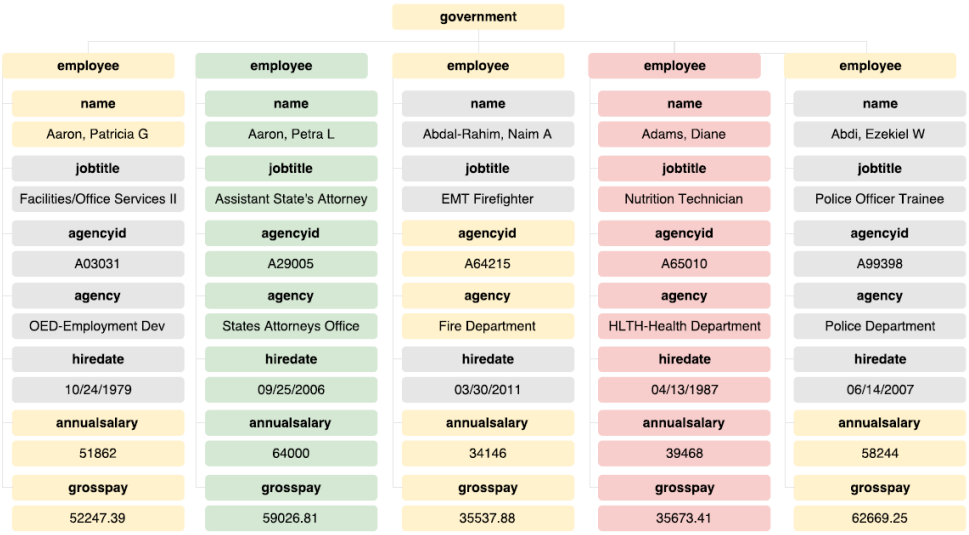


Figure 2: XML document related to Baltimore's City Hall employee registration at its second version (v2) using color markings to show the differences between v1 and v2. Green represents new additions, red represents deletions, yellow represents changes, and gray represent unaltered data.

We can easily identify these changes after an analysis of the two versions of this small XML document fragment, even without the color notations used in Figure 2. However, in the case of a company with a considerable number of employees, this analysis is not trivial. For example, the original document of the City of Baltimore (version *v1*) contains 13,966 employees. To keep up with changes made in XML documents, we need a method that detects the exact differences between two versions of the document, that is, what has been changed from one version to another in terms of the structures that compose an XML document (i.e., elements and attributes).

Additionally, in documents with large amounts of data, such as this sample document from the City of Baltimore, the identification of syntactic differences may not be enough to help users understand what changed, since the number of changes may be overwhelming. Instead, a semantic diff would be more helpful to identify, for example, that *Aaron, Petra L* was hired, that *Adams, Diane* was fired, and *Aaron, Patricia G* had a salary raise, among other changes.

3. XChange: Semantic DIFF of XML Documents

This section provides an overview of XChange, an approach to support the understanding of XML document evolution based on inference. The purpose of XChange is to enable the user to identify and understand semantic changes when analyzing versions of an XML document. Unlike existing approaches, XChange uses syntactic changes and a set of rules to infer the reason for the changes and support semantic diff.

Figure 3 presents an overview of XChange. At first, a domain expert must configure the tool for the knowledge domain of the XML document so that XChange can compute the semantic diff between two sequential (not necessarily consecutive) versions of an XML document. However, this configuration is done just once for each domain. After that, the end-user can use XChange multiple times to compare versions of different XML documents that belong to that domain. Figure 3 shows two distinct roles that interact with XChange: the *domain expert*, who does the XChange configuration for a given knowledge domain, and the *end-user* (called *user* from this point on) that compares the versions of the documents belonging to that domain.

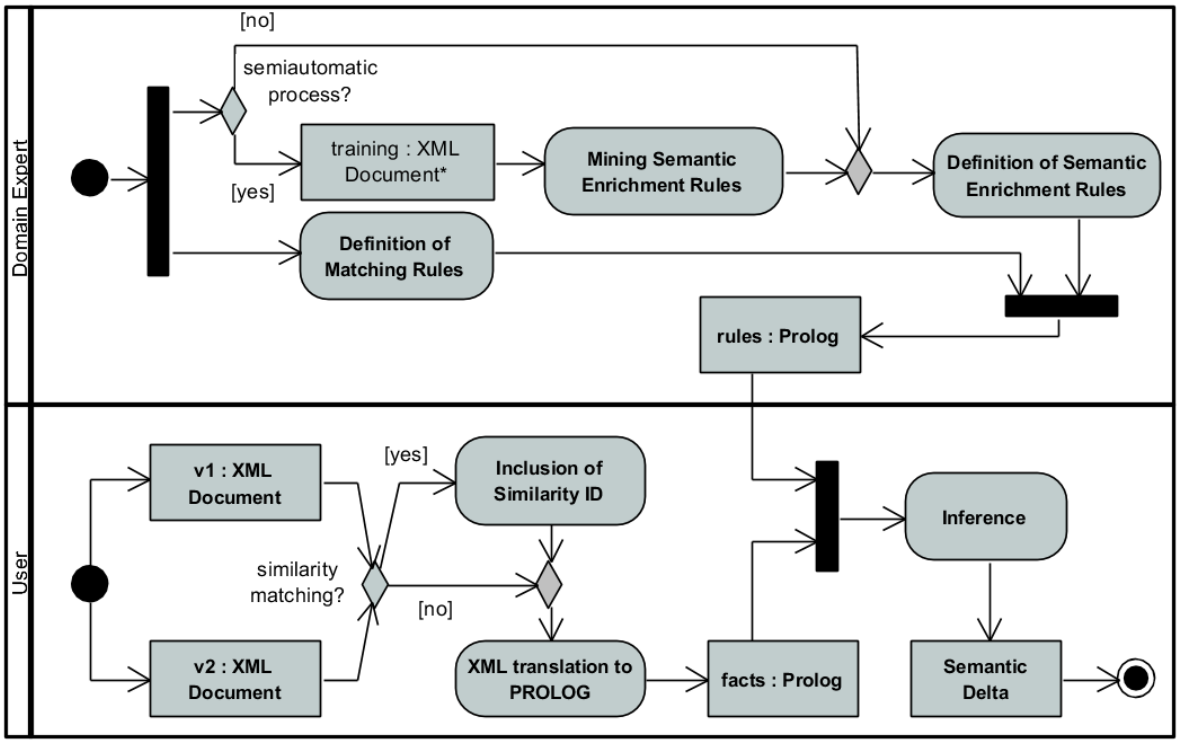


Figure 3: UML activity diagram with an overview of XChange

The domain expert must perform two main activities to configure XChange: (1) to define matching rules and (2) to define semantic enrichment rules. The **Definition of Matching Rules** (Section 3.1) aims at creating rules for identifying the corresponding elements in two sequential versions of an XML document. In this paper, we consider only two ways to identify matching elements in XML documents: *matching by key* and *matching by similarity*. A *key matching* rule could, for example, indicate that elements of two versions of an XML document match when one (or more) of their sub-elements have equal values (e.g., same Social Security Number). The *similarity matching* rule uses artificial IDs, which are obtained from the analysis of the similarity between the elements of the XML document versions, to indicate a matching. We provide more details of these matching rules in Section 3.1.

The second activity is the **Definition of Semantic Enrichment Rules** (Section 3.2). The domain expert can define these rules either manually or by using a semiautomatic process. The simpler manual rules could only indicate whether the document has changed or not. However, the domain expert can also define more elaborate manual rules, where several modifications are grouped into a single semantic change. For example, a rule might indicate that an employee was promoted by considering the syntactic information that she had her salary increased, and her job title changed.

Considering that the manual rule-making process can become complex even for a domain expert, XChange also provides a semiautomatic process for generating semantic enrichment rules. This semiautomatic process starts by **Mining Semantic Enrichment Rules** (Section 3.2.2) from a set of XML documents. The purpose of this mining step is to discover elements of the XML document that frequently change together when analyzing sequential versions. For example, if a salary increase often occurs together with a change in the job title, then the mining process would identify this joint occurrence, and the domain expert needs only to provide a meaningful name for this rule. XChange uses these enrichment rules, along with matching rules, during the inference stage to produce the semantic *diff*.

After the rules are configured for a given domain, an end-user can compare two versions of an XML document belonging to that domain. If the domain expert configures XChange to match elements by similarity, then XChange performs the **Inclusion of *Similarity ID*** in the versions of the XML document. We use the *Phoenix*

approach [20,43] to match elements across versions because it is more efficient than the state of the art approaches [20]. However, XChange is generic and can work with any other similarity algorithm.

Phoenix calculates the similarity between two versions *v1* and *v2* of an XML document, which ranges from 0 (0% – fully dissimilar) to 1 (100% – equal documents). It calculates this value recursively by comparing elements with the same parent in *v1* and *v2*. For each pair of elements, *Phoenix* considers the similarity between their names, their attributes, their textual contents, and their sub-elements. Two elements match when their similarity is greater than a previously configured threshold. XChange then artificially inserts a *Similarity ID* in the elements of *v1* and *v2* to indicate the match. Then, it occurs the **XML translation to PROLOG** (Section 3.3) by transforming each element of the two versions of the XML document into Prolog facts [44]. For this step, we needed a language that provides the innate capability to make inferences, such as Datalog[3] [45], RuleML[4] [47], or Prolog [48]. We adopted Prolog in XChange. In fact, Prolog has already been successfully used to perform queries with inference to XML documents [44,49].

Finally, XChange does the diff processing via **Inference** (Section 3.4). Our tool constructs the Prolog knowledge base from the facts and rules generated in the previous steps. XChange then automatically queries the knowledge base with the heads of each semantic enrichment rule. These queries provide as a response a semantic *delta* containing the elements of the XML document that fit into each of the situations modeled by the rules. In other words, this *delta* corresponds to the reason for the evolution of the XML document, from the former version to the later one.

3.1. Definition of Matching Rules

As mentioned earlier, there are different, XChange provides two different ways of identifying matching elements in XML documents: *matching by key* and *matching by similarity*. The simplest way to establish the correspondence between elements is the adoption of a key. A key matching rule specified in Prolog is shown in Figure 4. This rule uses the <name> element as an identifier (i.e., key) to match the elements of the versions of the Baltimore City document introduced in Section 2. This strategy guarantees the uniqueness of the <employee> element and can also be used in other scenarios/domains (in this case, with a different key, depending on the domain).

```
1 Match(EMPLOYEEBefore, EMPLOYEEAfter, NAME):
2   employee(before,EMPLOYEEBefore), employee(after,EMPLOYEEAfter),
3   name(EMPLOYEEBefore, _, NAME), name(EMPLOYEEAfter, _, NAME).
```

Figure 4: Matching rule by key

The user uses the matching rule to identify matching elements between two versions, *v1* and *v2*, for example. However, there is no guarantee that the key value remains the same between sequential versions depending on how users manage the XML documents. For example, a typo might have occurred in the value of <name> in *v1*, and someone fixed it in *v2*. Another related problem is that most XML documents do not have a key element or an associated schema [38–40]. XChange can use the artificial similarity ID produced by a preprocessing step that performs matching by similarity to solve this problem. Thus, the matching rule presented in Figure 4 can be used with the artificially generated identifier as a key, as shown in Figure 5, to establish the corresponding elements between the two versions of an XML document. For instance, if two employees have identical names, then the match by key will still be able to differentiate them. If we instead do a match by similarity, then Phoenix's similarity algorithm, which uses the Hungarian algorithm under the hood, will still be able to differentiate these employees since they probably have different attributes, such as mailing address, telephone, etc.

Another possible situation would be name changes due to marriage or divorce. As discussed before, for this situation the matching by similarity would still be able to match the employee before and after the name change due to other unchanged attributes. Similarly, another possible case would be a name change for one employee

[3] Datalog is a nonprocedural query language based on Prolog that does not allow complex terms as predicate arguments and have restrictions on the use of negation and recursion, making it incompatible with XChange.

[4] RuleML language was the result of an effort to provide a rule-setting pattern on the *Web* and describes both the information and its relationships, thus allowing inferences. It is a markup language aimed at representing *Web*-based inference rules, using Datalog as the inference mechanism.

(employee_#1) due to a marriage/divorce situation that results in the same name as another existing employee (employee_#2). Again, the matching by similarity would correctly match the employees since employee_#1 and employee_#2 have other attribute values that would remain unchanged. However, one way to get an incorrect match would be to change other attributes, besides the name, in such way that the similarity between versions for employee_#1 and employee_#2 are tied or employee_#1 attributes changed so much that employee_#2 now have a greater similarity score with the previous version of employee_#1. However, this case would probably be exceedingly rare to happen and the match by key would also prevent this mismatch.

```
1 match(EMPLOYEEBefore, EMPLOYEEAfter, XID):-
2   employee(before,EMPLOYEEBefore),employee(after,EMPLOYEEAfter),
3   xchangeid(EMPLOYEEBefore, _, XID),xchangeid(EMPLOYEEAfter, _, XID).
```

Figure 5: Matching rule by similarity with the use of an artificial identifier

3.2. Definition of Semantic Enrichment Rules

The domain expert defines the semantic enrichment rules once for each domain. These rules are later used each time the user needs to diff two versions of an XML document. The domain expert is fundamental in this process of setting the rules, which he/she can do manually, as described in 3.2.1, or using a semiautomatic process, as described in Section 3.2.2.

3.2.1. Manually definition of Semantic Enrichment Rules

As an example of a manually defined rule, Figure 6 presents some rules defined in the context of the Baltimore City Hall document introduced in Section 2. The *salary_increased* rule (lines 1-6) identifies employees who received a salary increase (<*annualsalary*>). The transferred rule (lines 7-12) identifies employees who have changed agencies (<*agencyid*>) while the fired rule (lines 13-18) matches to employees who were dismissed. The promoted rule (lines 19-27) identifies employees who received salary increases (<*annualsalary*>) and, also, changed roles (<*jobtitle*>), which means they were promoted. Finally, the *promoted_transferred* rule (lines 28-39) identifies employees who were both promoted and transferred. Note that it is also possible to define rules that are equivalent to a syntactic *diff*, which indicates that someone modified some element, as it is the case of the *salary_increased* rule. However, the domain expert can also define more elaborate rules that groups several syntactic modifications into a single semantic operation, as is the case of the *promoted* rule. We say that these rules produce a semantic diff because they use inference and seek to understand the meaning (semantics) of these changes.

```
1  salary_increased(NAME):-
2     match(EMPLOYEEBefore, EMPLOYEEAfter, XID),
3     name(EMPLOYEEBefore, _, NAME),
4     annualsalary(EMPLOYEEBefore, _, ANNUALSALARYBefore),
5     annualsalary(EMPLOYEEAfter, _, ANNUALSALARYAfter),
6     ANNUALSALARYBefore<ANNUALSALARYAfter.
7
8  transferred(NAME):-
9     match(EMPLOYEEBefore, EMPLOYEEAfter, XID),
10    name(EMPLOYEEBefore, _, NAME),
11    agencyid(EMPLOYEEBefore, _,AGENCYIDBefore),
12    agencyid(EMPLOYEEAfter, _,AGENCYIDAfter),
13    AGENCYIDBefore\=AGENCYIDAfter.
14
15 fired(NAME):-
16    employee(before,EMPLOYEEBefore),
17    xchangeid(EMPLOYEEBefore, _, XID),
18    name(EMPLOYEEBefore, _, NAME),
19    not((employee(after,EMPLOYEEAfter),
20    xchangeid(EMPLOYEEAfter, _, XID))).
21
22 promoted(NAME):-
23    match(EMPLOYEEBefore, EMPLOYEEAfter, XID),
24    name(EMPLOYEEBefore, _, NAME),
25    jobtitle(EMPLOYEEBefore, _, JOBTITLEBefore),
26    jobtitle(EMPLOYEEAfter, _, JOBTITLEAfter),
27    JOBTITLEBefore\=JOBTITLEAfter,
28    annualsalary(EMPLOYEEBefore, _, ANNUALSALARYBefore),
29    annualsalary(EMPLOYEEAfter, _, ANNUALSALARYAfter),
30    ANNUALSALARYBefore<ANNUALSALARYAfter.
31
32 promoted_transferred(NAME):-
33    match(EMPLOYEEBefore, EMPLOYEEAfter, XID),
34    name(EMPLOYEEBefore, _, NAME),
35    jobtitle(EMPLOYEEBefore, _, JOBTITLEBefore),
36    jobtitle(EMPLOYEEAfter, _, JOBTITLEAfter),
37    JOBTITLEBefore\=JOBTITLEAfter,
38    agencyid(EMPLOYEEBefore, _,AGENCYIDBefore),
39    agencyid(EMPLOYEEAfter, _,AGENCYIDAfter),
40    AGENCYIDBefore\=AGENCYIDAfter,
41    annualsalary(EMPLOYEEBefore, _, ANNUALSALARYBefore),
42    annualsalary(EMPLOYEEAfter, _, ANNUALSALARYAfter),
43    ANNUALSALARYBefore<ANNUALSALARYAfter.
```

Figure 6: Examples of semantic enrichment rules

XChange provides a graphical interface that helps in the creation of rules, easing the job of the domain expert. Figure 7 shows how this interface could be used to create the *promoted_transferred* semantic enrichment rule. Initially, the domain expert informs the *Rule Name*, the *Output*, and the *Conditions*. For the *Conditions*, XChange provides combo boxes with all the element and attribute names that appear in the document. In this example, the rule name is *promoted_transferred*, and the expected output involves the employee's name (<*name*>). The *condition*, in this example, is that the job title (<*job-title*>) of the employee in both versions must be different as well as the agency (<*agencyid*>) in which he/she works. Also, the annual salary (<*annualsalary*>) should be higher in the second version. Using this configuration, XChange generates the *promoted_transferred* Prolog rule, as shown in Figure 6 (lines 32-43).

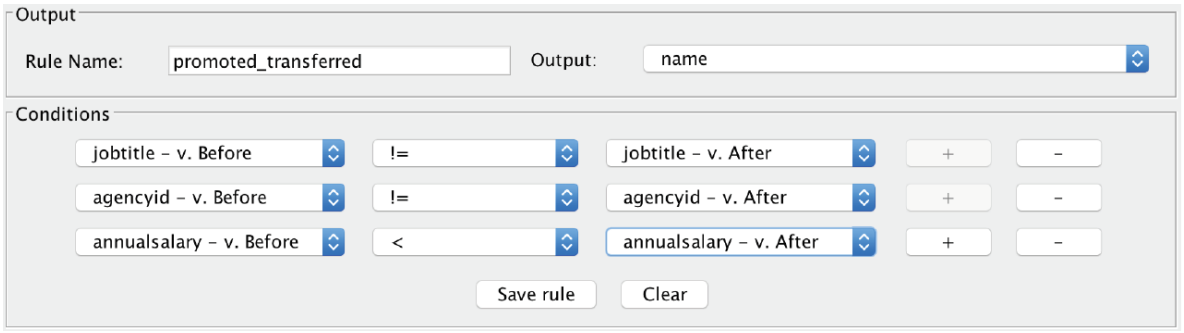


Figure 7: Interface to support the definition of semantic enrichment rules.

3.2.2. Mining Semantic Enrichment Rules

Defining rules manually may be a complex and time-consuming task. Therefore, XChange provides semiautomatic support for the construction of semantic enrichment rules based on frequent *itemsets* mining [50].

The goal of the semiautomatic support in XChange is to discover elements of the XML document that change together frequently when analyzing sequential versions and allow the domain expert to build the rules based on the presented suggestions. Thus, the association rules mining technique [51] was chosen for the identification of frequent *itemsets*. We selected the Apriori algorithm [50] and used the Weka [52] mining tool. The process occurs as described below.

Our approach composes the input (training) data for mining from sequential versions of an XML document. XChange applies the syntactic diff algorithm to each pair of XML document versions to generate *deltas*, which inform which elements have changed from one version to another. Our interest is in identifying the altered elements and the way they changed (for example, if a given element has had its value increased or reduced). Our approach generates a single consolidated *delta* from the diff of these versions. Next, our approach does a preprocessing of the output (consolidated delta) so that the Apriori algorithm can use it.

The Apriori algorithm returns the identified frequent *itemsets*, as exemplified in Table 1. In this table, the value associated with *y* (*yes*) indicates that an element of an employee changed from one version to another. The values associated with *u* (*up*) and *d* (*down*) indicate that an element changed to a higher or lower value, respectively, from one version to another. Row 3, for example, indicates that it is common to change *jobtitle* in conjunction with *annualsalary*, which indicates, for example, that some employees have been promoted (promoted rule defined in Figure 6). The *Support* column shows the number of times the Apriori algorithm identified the *itemset* in a total of 1,713 transactions.

Table 1: Examples of frequent *itemsets* found through data mining

	Changed Elements	Support
1	*agencyid=y annualsalary=u*	173
2	*jobtitle=y annualsalary=u grosspay=u*	182
3	*jobtitle=y annualsalary=u*	189
4	*agencyid=y grosspay=u*	190
5	*jobtitle=y grosspay=u*	195
6	*jobtitle=y*	209
7	*agency=y grosspay=d*	252
8	*agencyid=y agency=y*	257
9	*grosspay=d*	269
10	*agencyid=y*	292
11	*agency=y annualsalary=u grosspay=u*	796
12	*agency=y annualsalary=u*	931
13	*agency=y grosspay=u*	1009
14	*annualsalary=u grosspay=u*	1125
15	*annualsalary=u*	1269
16	*agency=y*	1322
17	*grosspay=u*	1383

After this step, our tool passes these suggestions on to the domain expert. The suggestions are displayed in the graphical interface presented in Figure 8. The domain expert analyzes the frequent *itemsets* and verifies if these suggestions have any relevant meaning. If so, the domain expert names the suggestions, giving a meaning to this joint change, as illustrated in Figure 8 with the *promoted* rule (Figure 6 lines 22-30).

Once the domain expert defines the semantic enrichment rules, either manually or by using this semiautomatic support, XChange can use these rules in the semantic diff of any version pair of an XML document from this knowledge domain.

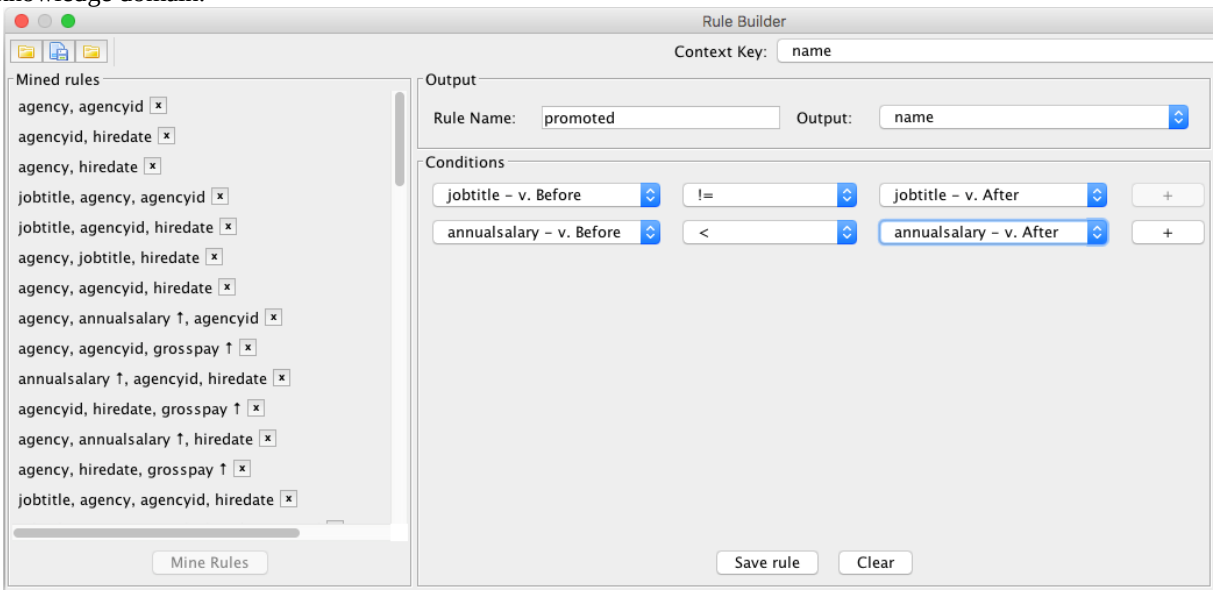


Figure 8: Suggestions for the definition of semantic enrichment rules using frequent *itemsets*

3.3. XML translation to PROLOG

We use the XML to Prolog translation method proposed by Lima et al. [44], which translates a single XML document to Prolog facts, transforming elements into predicates and their contents into constants. Since XChange deals with two documents at a time, XChange extends this translation method to use two versions of an XML document (in this case, *before* and *after*) as input and generate the corresponding Prolog facts. In the Baltimore City example, excerpts of the translation of *v1* and *v2* (Figure 9) are shown in Figure 10 to illustrate this process. Our approach performs the translation of the example in three steps described below: (1) **root translation**, (2) **complex elements translation,** and (3) **simple elements without attributes translation**.

The first step translates the **root** of v_1 *<government>* (line 1 of Figure 9) into a fact with the element name and a unique argument equal to an automatically generated identifier that is used to establish the link with its child elements), as shown in Figure 10(a) at line 1. Note that this is the only modification we made to the original translation method [44]. In the original method, the ID is a sequential number. In our modification, the root ID is the name of the version (we use *before* and *after* as names for the two versions that are being compared).

The second step is the **translation of the complex elements**, that is, those that have other elements as children, such as *<employee>* in Figure 9, line 2. Our tool creates a new identifier to relate the children to their parent. The result is the generation of a fact with the name of the element and two arguments: the parent identifier and a newly generated identifier to reference it, such as *employee(before, 2)* in Figure 10(a), line 2. Note that, in this case, *before* is the parent identifier.

The third step **translates the simple elements without attributes**. We show an example in line 3 of Figure 9 (*<name> Berube, Leslie A </ name>*). The result is a fact with a name equal to the element name and arguments equal to the identifier of the parent element, the identifier of the current element, and the content of the current element, as shown in line 3 of Figure 10(a) (*name (2, 3, 'Berube, Leslie A')*).

Documents that have other types of elements (such as mixed elements or simple elements with attributes) are translated following the method proposed by Lima et al. [44].

```
<government>
 <employee>
 <name>Berube,Leslie A</name>
 <jobtitle>COORDINATOR</jobtitle>
 <agencyid>A50701</agencyid>
 <agency>DPW-Water </agency>
 <hiredate>1978-06-26</hiredate>
 <annualsalary>50981</annualsalary>
 <grosspay>48956.35</grosspay>
 </employee>
 <employee>
 <name>Bond,Filishia M</name>
 <jobtitle>PARALEGAL</jobtitle>
 <agencyid>A06019</agencyid>
 <agency>Housing Com</agency>
 <hiredate>2001-06-25</hiredate>
 <annualsalary>50364</annualsalary>
 <grosspay>44941.01</grosspay>
 </employee>
 <employee>
 <name>Bailowitz,Anne</name>
 <jobtitle>EXECUTIVE</jobtitle>
 <agencyid>A65527</agencyid>
 <agency>HLTH-Health Dept</agency>
 <hiredate>2001-02-26T00:00:00</hiredate>
 <annualsalary>119000</annualsalary>
 <grosspay>103290.62</grosspay>
 </employee>
</government>
```

(a) ***Version v1.xml***

```
<government>
 <employee>
 <name>Blow,Teresa L</name>
 <jobtitle>MOTOR DRIVER</jobtitle>
 <agencyid>B49330</agencyid>
 <agency>TRANS-Highways</agency>
 <hiredate>2004-06-14</hiredate>
 <annualsalary>30742</annualsalary>
 <grosspay>31222.54</grosspay>
 </employee>
 <employee>
 <name>Berube,Leslie A</name>
 <jobtitle>ASSISTANT</jobtitle>
 <agencyid>A49101</agencyid>
 <agency>TRANS-Highways </agency>
 <hiredate>1978-06-26T00:00:00</hiredate>
 <annualsalary>55811</annualsalary>
 <grosspay>56025.54</grosspay>
 </employee>
 <employee>
 <name>Barnes,Ikea T</name>
 <jobtitle>AIDE BLUE CHIP</jobtitle>
 <agencyid>W02235</agencyid>
 <agency>Youth Summer </agency>
 <hiredate>2010-06-03T00:00:00</hiredate>
 <annualsalary>11310</annualsalary>
 <grosspay>1051.25</grosspay>
 </employee>
 <employee>
 <name>Bond,Filishia M</name>
 <jobtitle>EXECUTIVE</jobtitle>
 <agencyid>A06019</agencyid>
 <agency>Housing Com</agency>
 <hiredate>2001-06-25</hiredate>
 <annualsalary>52912</annualsalary>
 <grosspay>53047.47</grosspay>
 </employee>
</government>
```

(b) ***Version v2.xml***

Figure 9: Two different versions of the Baltimore XML file

```
government(before).
employee(before, 2).
name(2, 3, 'Berube,Leslie A').
jobtitle(2, 5, 'COORDINATOR').
agencyid(2, 7, 'A50701').
agency(2, 9, 'DPW-Water ').
hiredate(2, 11, '1978-06-26').
annualsalary(2, 13, 50981.0).
grosspay(2, 15, 48956.35).
employee(before, 17).
name(17, 18, 'Bond,Filishia M').
jobtitle(17, 20, 'PARALEGAL').
agencyid(17, 22, 'A06019').
agency(17, 24, 'Housing Com').
hiredate(17, 26, '2001-06-25').
annualsalary(17, 28, 50364.0).
grosspay(17, 30, 44941.01).
employee(before, 32).
name(32, 33, 'Bailowitz,Anne').
jobtitle(32, 35, 'EXECUTIVE').
agencyid(32, 37, 'A65527').
agency(32, 39, 'HLTH-Health Dept').
hiredate(32, 41, '2001-02-26T00:00:00').
annualsalary(32, 43, 119000.0).
grosspay(32, 45, 103290.62).
```

(a) *v1.pl*

```
government(after).
employee(after, 48).
name(48, 49, 'Blow,Teresa L').
jobtitle(48, 51, 'MOTOR DRIVER').
agencyid(48, 53, 'B49330').
agency(48, 55, 'TRANS-Highways').
hiredate(48, 57, '2004-06-14').
annualsalary(48, 59, 30742.0).
grosspay(48, 61, 31222.54).
employee(after, 63).
name(63, 64, 'Berube,Leslie A').
jobtitle(63, 66, 'ASSISTANT').
agencyid(63, 68, 'A49101').
agency(63, 70, 'TRANS-Highways ').
hiredate(63, 72, '1978-06-
26T00:00:00').
annualsalary(63, 74, 55811.0).
grosspay(63, 76, 56025.54).
employee(after, 78).
name(78, 79, 'Barnes,Ikea T').
jobtitle(78, 81, 'AIDE BLUE CHIP').
agencyid(78, 83, 'W02235').
agency(78, 85, 'Youth Summer ').
hiredate(78, 87, '2010-06-
03T00:00:00').
annualsalary(78, 89, 11310.0).
grosspay(78, 91, 1051.25).
employee(after, 93).
name(93, 94, 'Bond,Filishia M').
jobtitle(93, 96, 'EXECUTIVE').
agencyid(93, 98, 'A06019').
agency(93, 100, 'Housing Com').
hiredate(93, 102, '2001-06-25').
annualsalary(93, 104, 52912.0).
grosspay(93, 106, 53047.47).
```

(b) *v2.pl*

Figure 10: Prolog facts generated from versions v1 and v2 of Figure 9

3.4. Inference

The **Inference** step is responsible for the semantic diff processing. Our tool constructs the knowledge base with the generated Prolog facts that represent the versions of the XML document, the semantic enrichment rules, and the matching rules generated in the previous steps. Then, queries are submitted to the knowledge base using the heads of each semantic enrichment rule, which provides as answer the XML elements that were subject to semantic changes. These elements are then used to build an XML-formated semantic delta, as shown in Figure 11. In other words, this delta corresponds to the reason for the evolution of the XML document, from a former version to a later one.

```
<diff-set from="v1 - Wed Nov 04 16:51:53" to="v2 - Fri Nov 04 16:51:52">
 <diff name="salary_increased">
  <description>
   <change attr="annualsalary" type="increased"/>
  </description>
  <delta count="2" annualsalary="7378">
   <employee name="Berube,Leslie A">
    <annualsalary before="50981" after="55811" delta="4830"/>
   </employee>
   <employee name="Bond,Filishia M">
    <annualsalary before="50364" after="52912" delta="2548"/>
   </employee>
  </delta>
 </diff>
 <diff name="fired">
  <description>
   <change attr="name" type="deleted"/>
  </description>
  <delta count="1" annualsalary="-119000" grosspay="-103290.62">
   <employee name="Bailowitz,Anne">
    <jobtitle>EXECUTIVE</jobtitle>
    <agencyid>A65527</agencyid>
    <agency>HLTH-Health Dept</agency>
    <hiredate>2001-02-26T00:00:00</hiredate>
    <annualsalary>119000</annualsalary>
    <grosspay>103290.62</grosspay>
   </employee>
  </delta>
 </diff>
 <diff name="promoted">
  <description>
   <change attr="jobtitle" type="different"/>
   <change attr="annualsalary" type="increased"/>
  </description>
  <delta count="2" annualsalary="7378">
   <employee name="Berube,Leslie A">
    <jobtitle before="COORDINATOR" after="ASSISTANT"/>
    <annualsalary before="50981" after="55811" delta="4830"/>
   </employee>
   <employee name="Bond,Filishia M">
    <jobtitle before="PARALEGAL" after="EXECUTIVE"/>
    <annualsalary before="50364" after="52912" delta="2548"/>
   </employee>
  </delta>
 </diff>
 <diff name="promoted_transferred">
  <description>
   <change attr="jobtitle" type="different"/>
   <change attr="agencyid" type="different"/>
   <change attr="annualsalary" type="increased"/>
  </description>
  <delta count="1" annualsalary="4830">
   <employee name="Berube,Leslie A">
    <jobtitle before="COORDINATOR" after="ASSISTANT"/>
    <agencyid before="A50701" after="A49101"/>
    <annualsalary before="50981" after="55811" delta="4830"/>
   </employee>
  </delta>
 </diff>
 ...
</diff-set>
```

Figure 11: Semantic delta generated by XChange from applying the Prolog rules shown in Figure 6 on the facts shown in Figure 10

Figure 11 shows that *Berube, Leslie A* (line 7), and *Bond, Filishia M* (line 10) received a salary increase (line 2). Also, the same two employees (lines 36 and 40) were promoted (line 30), but only the employee *Berube, Leslie A* (line 53) was promoted and transferred (line 46). Moreover, the *delta* provides some data summaries. Line 6, for example, shows the total number of employees identified by the rule (count = 2), as well as the sum of the salary increases granted (*annualsalary = "7378"*). Line 8 shows the salary of the employee in the two analyzed versions and the difference between these values (attribute *delta*).

4. Experimental Study: Comprehension of XML document evolution

We did not find any approaches that focus on the semantic diff to compare with XChange. However, syntactic diff approaches also aim at providing an understanding of the changes that someone made on an XML document. In previous work [20], we compared the quality of the delta produced by state-of-the-art diff algorithms [14,27]. The best approach in terms of recall was XDiff [27]. We thus chose XDiff and XChange to conduct an experimental study, aiming at comparing them concerning the **effectiveness** and **efficiency** of participants in the analysis and understanding of the evolution of XML documents.

We gave the participants a set of tasks that aimed at checking their understanding of the evolution of two XML documents. Participants conducted each set of tasks using a different tool (we provide more details on our experimental setting in Section 4.1). We calculate the **effectiveness** in terms of the number of correct answers obtained in each task, as discussed in Section 4.2.1. Complementarily, **efficiency** is calculated in terms of the total number of correct answers per minute, as discussed in Section 4.2.2. Thus, the objective of this experimental study is to answer the following research questions:

RQ1*: Does XChange's semantic change identification allow users to be more effective in understanding the XML document evolution when compared with using X-Diff's syntactic changes identification?*
RQ2*: Does XChange's semantic change identification allow users to be more efficient in understanding the XML document evolution when compared with using X-Diff's syntactic changes identification?*

4.1. Materials and Methods

We created a fictitious situation to facilitate the participant's immersion in the context of the study: The city of Baltimore hired the participant (user with experience in manipulating XML documents). His/her first task in the new job would be to analyze two versions (*v1* and *v2*) of the employees from the Baltimore's City Hall example, the resulting delta, and answer some questions.

We adopted the mining strategy presented in Section 3.2.2 to generate the semantic enrichment rules. To do so, we split each version of the XML document into fifteen fragments (named 0 to 14) using the <name> element as a parameter for the horizontal fragmentation. We then used the first fragment (Fragment #0) of each version in the mining step (

Table 2), which includes only employees with names starting with "A". To avoid bias due to subjective interpretations, we used all the 17 *itemsets* produced by the mining step as is (i.e., without being analyzed and labeled by a domain expert) and derived one rule for each itemset. Next, we used the rules over the remaining fragments to generate the semantic delta.

Table 2: Baltimore fragment #0 characteristics (size in KB)

Fragment	v1		v2		v3		v4		v5	
	#emp	size	#emp	size	#emp	size	#emp	size	#emp	size
0	446	121	513	140	647	180	628	170	471	133

We designed a short course about XML diff of approximately 30 minutes, aiming at introducing the subject to the participants. Also, we contextualized the experimental study by using an example task, like those that the participants would perform during the experimental study. We showed the participants how to accomplish that task using both tools (XChange and X-Diff).

We developed six tasks inspired by different types of SQL queries [53]. Since SQL is a powerful and largely adopted language to query data, using it as inspiration helped us to cross-examine the possible types of questions one can ask about two versions of XML documents. Table 3 presents the classification we used, and the task associated with each type. The columns of the table deal with the two possible types of results, analogously to the projection of a query (*SELECT clause*). In any query, the user can enumerate the results, or aggregate them (*max, min, count, sum*). The rows of Table 3 include selection types that the user can apply in a query. The first is existential, which compares two sets by selecting the results that appear in the first set but do not appear in the second (*minus* operator). The second line contemplates changes occurring in the same instance of the result. To do this, a union (of the corresponding elements) and selection (*WHERE clause*) are applied to matched elements (for example, $e_1.salary \neq e_2.salary$, where e_1 corresponds to an employee in the first version, and e_2 is the element that matched to e_1 in the second version). Finally, the third row of the table deals with constraints on the result set. As for the second row, the corresponding elements are joined, and then a *HAVING* constraint is applied to filter only the desired results (e.g., the employee with the highest annual salary increase).

Table 3: Classification of tasks

	Enumeration	**Aggregation**
Existential	Task 1: Which employees were dismissed?	Task 2: What is the financial impact of hiring and firing based on the annual salary?
Change	Task 3: Who was promoted, that is, had a gross salary increase and changed their position?	Task 4: How many employees have been transferred, that is, changed agency?
Content	Task 5: Which employee had the highest annual salary increase?	Task 6: What is the financial impact of gross wage increases?

To answer **RQ1**, we counted the number of correct answers. We also recorded the duration of each task of the experimental study at each stage to support the analysis of **RQ2**. Table 4 shows the dependent variables of this study. We used two treatments for the independent variable that refers to the diff of XML documents: (1) the syntactic diff produced by X-Diff and (2) the semantic diff produced by XChange. The participants of the experimental study were students and alumni of several courses offered by the *Computer Science department at the Federal University of Juiz de Fora* (*UFJF*), with experience in manipulating XML documents. We made sure those students were not taking any courses with the authors of this paper in the semester when the study was conducted.

Table 4: Design of the experimental study

Dependent Variables	Number of correct answers (effectiveness), Number of correct answers per minute (efficiency)
Independent Variable	XML document diff
Treatments	syntactic diff produced by X-Diff semantic diff produced by XChange
Context	real dataset
Dataset	Baltimore

We ran a pilot study with the same structure described in this planning, which we carried out with only two participants before the execution of the study. The goal was to detect possible problems in the study execution. Also, a computational environment was carefully prepared to isolate tasks and provide only the data and tools

needed by the participant at each stage of the study. The participants did not have access to the internet during the study.

We invited approximately 200 students and alumni of the *Federal University of Juiz de Fora* (*UFJF*). Sixty of them participated in the experimental study. The study was divided into five sessions to accommodate the availability of the participants and to make sure we could accurately monitor the participants in each session. All participants filled a consent form and a characterization form to participate in the study. Then, we presented the introductory course related to XML diff and the contextualization with an example task, as previously discussed.

The study participants, selected by convenience, are at the undergraduate level (mostly), master's, or doctoral level, as shown in Figure 12. All the participants have already studied the Database and Software Engineering disciplines. Figure 12 also presents the experience of the participants. Most of the participants are graduates or undergraduates, with a small group of participants having more than six years of industry experience while a significant group has no more than two years of experience in personal and academic projects. Most of the participants have industry experience related to Database and Version Control. Besides, about 25% have industry experience with XML and file diff. Most of the participants have experience in XML and diff only from books or from the attended courses.

We divided the participants into two groups (G1 and G2) for each of the five sessions. For that, we used the responses of the characterization questionnaire (summarized in Figure 12), aiming at dividing the participants in homogeneous groups for each session. We randomly divided the participants within each level of academic training, followed by the degree of XML experience, when necessary. We used the *Latin square* [54] design for both treatments (XChange and X-Diff). Our experimental study plan divided each session into two steps: **step 1** containing the first three tasks, and **step 2** containing the last three tasks. G1 used XChange, and G2 used the X-Diff tool to complete the tasks of **step 1**. During **step 2**, we switched the tools of each group. Thus, G1 used X-

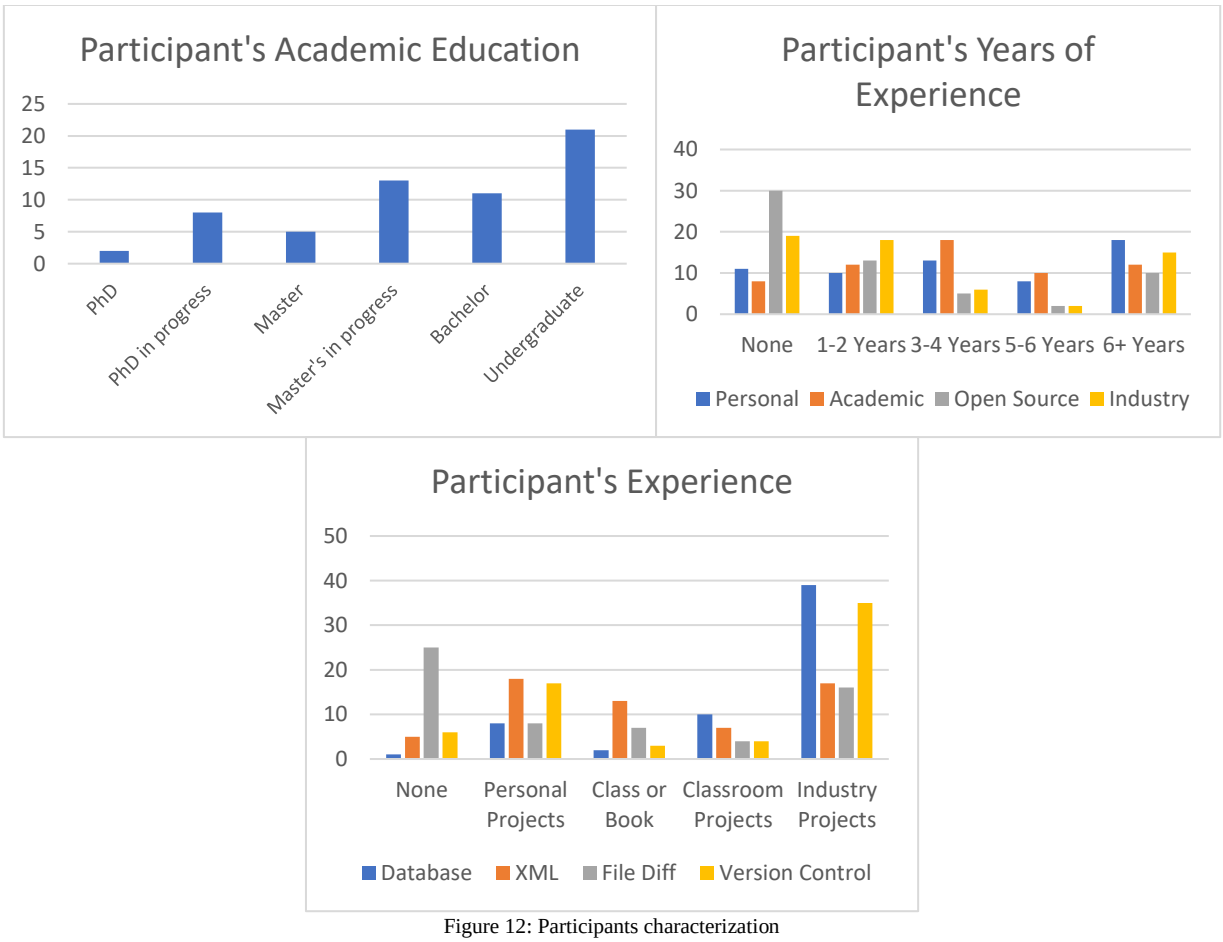


Figure 12: Participants characterization

Diff, and G2 used XChange to complete the tasks of **step 2**. We did not inform the participants of the existence and purpose of this division of tasks into groups, so as not to influence the execution of the tasks. We also did not inform the participants which tool we were proposing.

Participants received version v1 and version v2 so they could consult the documents during their analysis, if necessary, as well as the resulting delta from the used tool (X-Diff or XChange) to use in solving the tasks at each step. We advised the participants to use only the delta in the resolution of the tasks whenever possible. Finally, participants completed a follow-up questionnaire. The goal was to obtain qualitative information about the study, including the participants' perception of the XChange and X-Diff approaches.

4.2. Results and Discussion

We started by checking if our results followed a normal distribution. For this, we used the Shapiro-Wilk test [55] with a 95% confidence interval [56]. Both variables (*Number of Correct Answers* and *Duration*) had their normality assumption violated since their *p-values* were lesser than the *α-value*. Thus, this sample does not follow a normal distribution. Therefore, we used the non-parametric Mann-Whitney test for two independent samples [57] for the statistical analysis of the data[5]. Additionally, since we performed six tests (one for each task) analyzing the same hypotheses, we apply the Bonferroni correction to adjust the 0.05 target α-value accordingly. Compared to other corrections, the Bonferroni correction is the most pessimistic option leading to the smallest adjusted alpha-value [58], leading us to conservative significance confirmations. The Bonferroni correction, in our case, adjusts the alpha-value to 0.05/6 = 0.0083. The results are shown in Table 5.

Table 5: Cliff Delta and p-value for the number of Correct Answers (effectiveness) and Duration variables

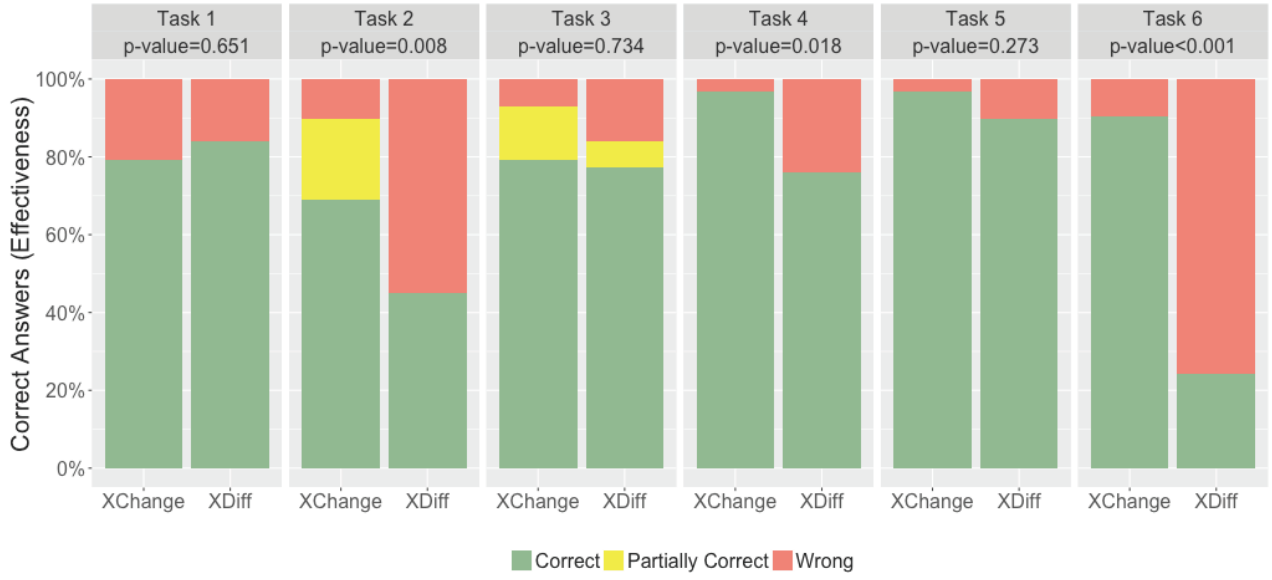
Task	**Correct Answer**		**Duration**	
	Cliff's Delta	***p-value***	***Cliff's Delta***	***p-value***
1	-0.045 (negligible)	*0.6510*	0.010 (negligible)	*0.9469*
2	0.351 (small)	***0.0081***	**-0.692 (large)**	***< 0.0001***
3	0.036 (negligible)	*0.7342*	0.219 (small)	*0.1449*
4	0.131 (negligible)	*0.0182*	0.089 (negligible)	*0.2768*
5	0.002 (negligible)	*0.2733*	**-0.764 (large)**	***< 0.0001***
6	**0.554 (large)**	***< 0.0001***	**-0.911 (large)**	***< 0.0001***

4.2.1. Effectiveness

Analyzing the results for the number of *Correct Answers* from Table 5 after applying the Bonferroni correction (α-*value* = 0.0083), tasks 2 and 6 have *p-values* lower than the adjusted α-*value*, which indicates a statistically significant difference between the scores of these tasks using XChange and X-Diff. For other tasks, we could not observe significant differences. In addition, we used Cliff's Delta ($|d|$) [59] effect size measure. Cliff's Delta is a non-parametric measure that allows quantifying the magnitude of the difference between two groups that do not meet the normality assumptions. We adopted the following interpretation to the Cliff's Delta values [60]: $|d| < 0.147$ as a negligible difference, $0.147 \leq |d| < 0.330$ as a small difference, $0.330 \leq |d| < 0.474$ as a medium difference, and $0.474 \leq |d|$ as a large difference. We highlighted in bold the values where the Cliff Delta points to a medium or large difference in Table 5.

The stacked bar graphs presented in Figure 13 show the *number of correct answers*, the mean, and standard deviation obtained in each task of the study. We considered an answer for task 2 as partially correct if it correctly identified the employees that were fired and hired but did not include the financial impact calculations, as

[5] Experiment data is available at https://dew-uff.github.io/xchange/



MEAN	0.79	0.83	0.86	0.45	0.92	0.82	0.96	0.75	0.96	0.89	0.90	0.24
STD	0.41	0.37	0.34	0.50	0.27	0.38	0.17	0.43	0.17	0.30	0.30	0.43

Figure 13: Analysis of the "Number of Correct Answers"

requested in this task. Similarly, we considered an answer for task 3 as partially correct if it missed identifying only one employee and all the identified ones were correct. It is worth noting that we considered partially correct answers in tasks 2 and 3. This graph complements the results obtained from the Mann-Whitney and Cliff Delta tests, as shown in Table 5. Figure 13 also shows that the XChange approach obtained a larger number of correct answers in all tasks with a statistically significant difference. The total number of correct answers of participants who used X-Diff was only greater than those who used XChange at task 1 but without a statistically significant difference.

RQ1. *Does XChange's semantic change identification allow users to be more effective in understanding the XML document evolution when compared with using X-Diff's syntactic changes identification?*

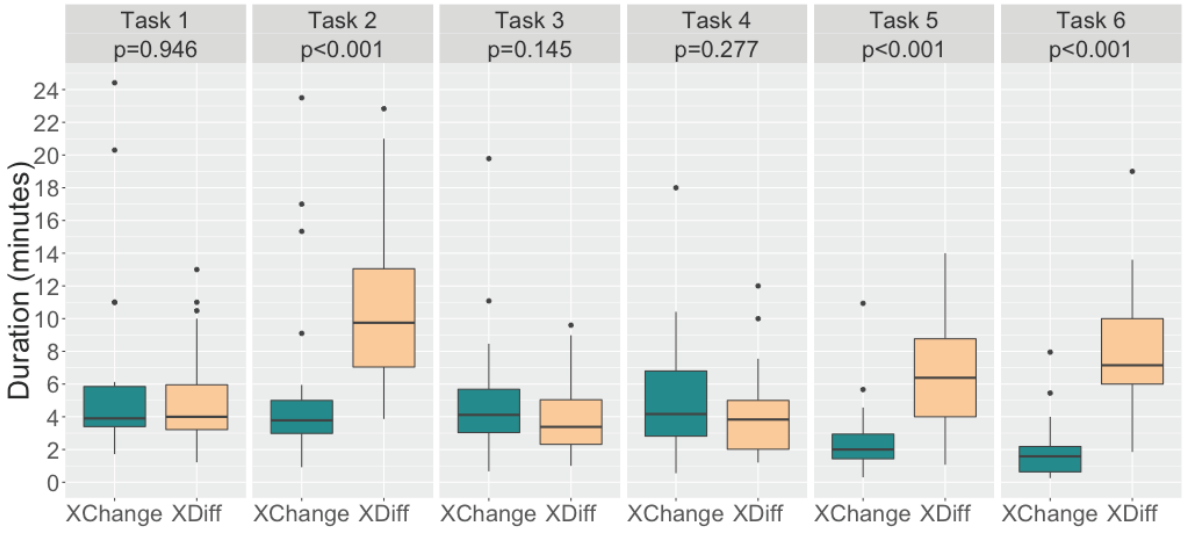
Answer: Yes. Participants who performed the tasks using XChange obtained the highest number of correct answers in all tasks, but task 1. However, differences in tasks 1, 3, 4, and 5 are not statistically significant. Only tasks 2 and 6 had statistically significant differences. As such, XChange can be more effective in understanding the evolution of XML documents when compared with X-Diff. Complementing the results, participants considered that the delta produced by XChange facilitates the analyses since it provides summarizations even though the overall resulting delta is larger than X-Diff. They mentioned that since they often needed to go through the whole document to calculate summarizations when using X-Diff and thus, they were more subject to errors or incomplete answers.

4.2.2. Efficiency

The boxplots presented in Figure 14 summarize the XChange and X-Diff distributions for the *duration* variable, showing significant duration differences for tasks 2, 5, and 6 (as also shown in Table 5) even when applying Bonferroni correction (α-*value* = 0.05/6 = 0.0083). Participants using XChange were faster to complete these tasks than participants who used X-Diff.

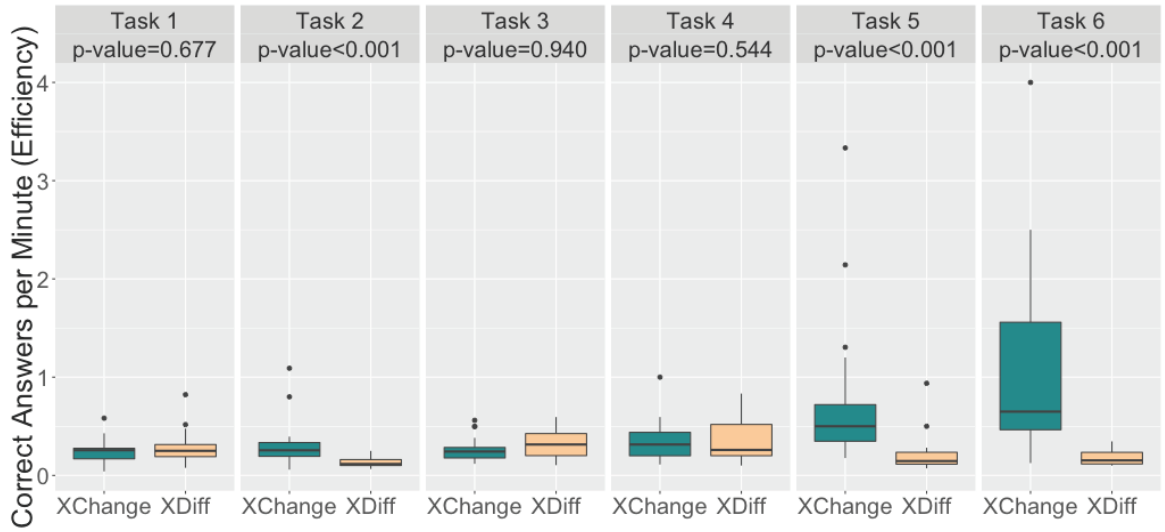
After analyzing the *duration* variable and the *number of correct answers* variable, we contrast the efficiency of both approaches by considering the total *number of correct answers* per minute (true positives per run time) as shown in Figure 15. X-Diff gets more *correct answers* per minute (considering the median) for tasks 1 and 3, while XChange gets the most *correct answers* per minute for tasks 2, 4, 5, and 6.

We ran the non-parametric Mann-Whitney hypothesis test and Cliff's Delta effect size for the results presented in Figure 15, obtaining *p-value* < α*-value* with Bonferroni correction (i.e. α*-value* = 0.0083) only for tasks 2, 5, and 6. The Cliff's Delta result for those tasks showed that their differences were considered large, as shown in Table 6. Therefore, there is a statistically significant difference between the scores for tasks 2, 5, and 6 using XChange and X-Diff. We can conclude that the identification of semantic changes used by XChange can make the understanding of the evolution of XML documents more efficient, based on the number of *correct answers* per minute than the identification of syntactic changes used by X-Diff. It is interesting to note that for task 5, the participants in both groups had similar results related to correct answers but, due to the aggregation nature of XChange's summary, they could quickly check all salary increases, without needing to run through the entire document, and easily point out the one that increased the most.



MEAN	5.76	4.98	5.31	10.45	5.1	4	5.08	4.1	2.45	6.62	1.86	7.8
STD	5.11	2.84	4.98	4.93	3.6	2.22	3.57	2.55	1.96	3.25	1.67	3.57

Figure 14: Analysis of the variable "duration."



MEAN	0.24	0.27	0.31	0.13	0.26	0.33	0.34	0.36	0.69	0.2	1.07	0.19
STD	0.12	0.15	0.24	0.05	0.12	0.15	0.19	0.22	0.64	0.17	0.92	0.09

Figure 15: Total number of correct answers per minute

Table 6: Cliff Delta and p-value for the number of Correct Answers per Minute (efficiency)

Task	Correct Answers per Minute	
	Cliff's Delta	***p-value***
1	0.912 (large)	0.6775
2	-0.912 (large)	**<0.0001**
3	-0.935 (large)	0.9399
4	-0.920 (large)	0.5443
5	-0.822 (large)	**<0.0001**
6	-0.948 (large)	**<0.0001**

RQ2. *Does XChange's semantic change identification allow users to be more efficient in understanding the XML document evolution when compared with using X-Diff's syntactic changes identification?*

Answer: Yes. The execution of the tasks by the participants using XChange generated more *correct answers* per minute, with a significant difference for tasks 2, 5, and 6. In other tasks, the differences are not statistically significant. Therefore, XChange is more efficient than X-Diff in understanding the evolution of XML documents. Furthermore, the participants mentioned in the follow-up questionnaire that the size of the delta resulting from the X-Diff was a positive point since it was smaller than XChange's. However, the participants had to go through the entire document to find the desired answers. The summary and format of the delta used by XChange, which aggregates data by semantic changes, helped participants to find the correct answers in a shorter time, without having to go through the entire document. Therefore, the participants could quickly do a search to find the desired semantic change and see all the changes that belong to that category in an aggregated list.

4.2.3. Intuitive Analysis of the Results

Task 1 required the participants to find all the employees that were fired. This is translated, in versioning, as finding all instances that were present at *v1* and absented at *v2*. For XChange, this was simply finding the fragment in the delta corresponding to *attr="name" type="deleted"* and check all the listed employees. For X-Diff, the participants were required to look in the entire delta to locate all employees with "*deleted*" instances or "*updated from*" in the employee name (i.e., changed name). The second case ("*updated from*") is due to X-Diff matching, which can match wrong employees between versions when hiring and firing occur at the same time (e.g., fired some employees and hired other employees) and mark them as an update. For example, in a situation where employee Valeria is fired and employee Carlos is hired and no other changes (no additional hiring or firing) happened, X-Diff matches employee Valeria from the previous version with employee Carlos from the new version and marks that Carlos was "*updated from*" Valeria. In both tools, this can be accomplished with the search operation, so there was no clear difference in efficiency and effectiveness.

Task 2 required the participants to determine the financial impact based on annual salary due to all the hiring and firing of employees between document versions. This task was similar to task 1, with the addition of also finding all new employees and calculating the sum of the annual salary of all employees that were hired minus the sum of the annual salary of all employees that were fired. In XChange, this was accomplished by finding the fragments related to change *attr=name type="inserted"* and *change attr="name" type="deleted"* and adding the listed annual salary from each fragment, which was already a sum of the annual salary of all employees that were inside that fragment. The X-Diff process is similar to the one from Task 1, but needed to include the "*INSERTION*" elements and the employees at the left side of the "*updated from*", since these were the ones that were hired. Then, manually computing the difference in annual salary when hiring and firing all those listed employees. Therefore, X-Diff required more steps than XChange to find all hired and fired employees and needed to read the entire diff file, which is more error-prone, and probably this is the reason for having more incorrect answers. Furthermore, this is an aggregation task, as pointed out in Table 3, and the grouped answer of XChange was probably the reason for being more efficient than X-Diff, since the participants needed to find all instances across the document and then do the math instead of having them all grouped together.

Task 3 concerned the participants identifying all the employees that had an increase in gross salary and also changed their job title. XChange has a single fragment for {*change attr=jobtitle type="different" AND change attr=annualsalary type="increased"*}. Thus, the participant only needed to look at the diff file until finding this fragment and list the employees that appeared in that fragment. Solving this task in X-Diff was done in a similar way: the participant needed to search the diff file for all instances that had "*updated from*" in both *jobtitle* and *annualsalary*. Like task 1, this could be accomplished with simple search operations and that is probably why it yielded similar results in both tools.

Task 4 asked participants to count all employees that were transferred. This is translated to finding employees that had agency change. This is like task 3, but instead of two arguments (*jobtitle* and *annualsalary*), we only have one argument change and need to count the number of instances instead of listing. Same as before, this task could be solved with simple search and count in the diff file from either approach.

Task 5 required finding the employee that had the highest increase in gross pay. Therefore, the participants were expected to find all employees that had changes in gross pay and get the one that had the biggest variation. XChange provides a fragment that lists all employees that had change *attr=grosspay type="different"*. Thus, the participant only needed to compare the results, that were closed together, and pick the one with the greatest positive change. X-Diff is done similarly, but the changes are scattered across the entire diff file instead of being grouped. This is probably the reason for XChange being more efficient than X-Diff in this task for finding the maximum gross pay change.

Task 6 required to determine the financial impact of gross wage increase. This is like the previous task with the difference being summing up all the instances that gross pay was increased instead of only finding the max. Furthermore, this is also an aggregation task, similar to task 2, as pointed out in Table 3. Thus, XChange nature of grouping semantically related changes was probably the reason for being more efficient than X-Diff and less error-prone.

All in all, the probable cause for wrong answers in XChange across all tasks could be because the participants found the incorrect fragment due to stopping at the first fragment that closely resembled what was asked. This would be analogous to stopping the search when finding the local maximum instead of trying to find the global maximum. The probable cause for wrong answers in X-Diff, on the other hand, could be the necessity of looking at the entire diff file to find all the requested instances, making it more error-prone due to missing or forgetting one instance or taking longer to do a mathematical computation when everything is scattered across the diff file.

4.3. Threats to Validity

We sought to avoid threats that could impact or limit the validity of the results [61] during the planning of this study. However, we cannot guarantee that such threats have not affected our results. Therefore, we describe in this section the threats identified in the context of this study.

Our study did not occur in a single day, but in five sessions distributed in four days, depending on the availability of the participants. This might have impacted the results since it is not possible to confirm that the circumstances were the same at each session. However, we used the same script and the same computational environment to minimize this threat.

The execution of the study consisted of two steps. Although we designed the study to avoid participants' learning effects, providing different tasks at each step, it is not possible to confirm that we have completely eliminated this side-effect.

Another threat to the study is related to the diff generation from XChange, which requires to use the rules defined manually by the domain expert or rules from the automatic mining-based method previously described. In the study, we only used rules generated from the semantic enrichment process through the automatic mining procedure to avoid external interference from domain experts. We used the minimum support of 0.03 to identify the frequent *itemsets* to generate these rules. This decision might negatively affect the XChange results due to the absence of the domain expert since he/she would provide meaningful names to the generated rules from the mining process.

The participants' understanding of the tasks during the study is related to how we elaborated the tasks. We tried to minimize this threat by analyzing all the supporting material in a pilot study to reduce this interference.

Since this is not an observation study, due to the reasonable number of participants, we assumed that the participants followed the instructions and the order of the activities in each task. However, to minimize this threat, we only presented the second step of each session after the participant completed the first step. In addition, another similar threat is related to the duration of each task. We did not monitor it due to the high number of participants in each session and expected the participant to correctly provide the amount of time taken to complete each task (task duration).

Another threat is related to the nature of using people during any study that requires the completion of several tasks since each participant has different problem-solving capabilities. We minimized this threat by using the characterization questionnaire answers for dividing the participants into a more homogeneous group in each session. Furthermore, the participants in this study are, for the most part, undergraduate students, which limits the representativeness of the people who could benefit from the approach. However, some of the students attended or are attending graduate courses, or have to experience in the industry, which serves to reduce this type of threat.

Some could argue that the sample size used in this experimental study is small and limited, making it a threat to the results, leading in some statistically inconclusive answers. However, 30 participants are considered as a sufficiently large sample for a controlled experiment [54]. Since we have 60 participants in our study, this increases the statistical validity of the conclusions we obtained.

Finally, the grouping of tasks by type tends to assist the analysis process of the data. However, although some of these tasks may have a higher degree of difficulty than others, we assigned the same weight to all tasks, which could influence the results. Due to the subjectivity in evaluating the degree of difficulty (which would introduce bias in the data analysis), we decided to maintain this setup.

5. Related Work

Several approaches in the literature [15,16,62,63] compare XML documents. However, such approaches focus on the **syntactic diff**, which is related to syntactic modifications in the documents. Hence, in the Baltimore's City Hall example, these approaches can detect, for example, changes in the value of an employee's salary. However, they do not convey the meaning of this change.

Some approaches to detect changes in web pages written in XML and HTML are also related to our proposed approach. WebVigiL [62] is a change tracking system for Web pages written in XML and HTML. The change detection module of this approach consists of two algorithms: CH-Diff and CX-Diff. CH-Diff is an algorithm for detecting changes in HTML documents. CX-Diff [15,16], on the other hand, is a specific algorithm for detecting XML document changes. The user can specify, for example, the page he/she wants to monitor, the type of change, and how the tool notifies the changes. However, this approach can overload the servers while computing the delta due to the highly expensive computational cost of the algorithms.

Some diff approaches [12,14,20,21,25,27] are based on a **structural analysis** of XML documents. Their main strategy is to find and match fragments of data in both versions of an XML document. After that, they focus on identifying the correct order of operations that transforms one version of the XML document into another, independently of the domain. XyDiff [14] is one such approach that detects the differences between the versions of an XML document from a hierarchical tree-based approach. XyDiff uses the XyDelta format [19], a single XML file containing all the detected differences. By using hashes, XyDiff removes identical subtrees from the comparison, thus reducing the amount of data to compare, which provides better performance when compared with others. XyDiff presents the delta as a list of operations (*insert*, *delete*, *update*, and *move*). This output format also easy the mapping of the delta to another format. Conversely, from the user perspective, it is more difficult to identify the differences between the versions.

X-Diff [27] uses unordered trees to detect differences between versions of an XML document. It focuses on guaranteeing the minimum delta. The algorithm detects the minimum mapping between the children of two subtrees, reducing the problem to a maximum flow problem with minimal cost. When it comes to large XML documents, its execution time is long because X-Diff finds the minimal delta in quadratic time. Unlike XyDiff, X-Diff does not use a list of operations to represent the delta. X-Diff uses its own version of the XML document to record the differences. Furthermore, this strategy of representing differences hardens the reconstruction of the

former version from the later version and the delta. X-Diff only considers the standard operations (*insert*, *delete*, and *update*).

XRel_Change_SQL [25] detects differences between versions of an XML document stored in a relational database. This approach uses SQL queries to calculate the *diff*. Like X-Diff, it is based on an unordered tree model. However, it considers the standard operations (*insert*, *delete*, and *update*) in addition to the *move* operation, which detects changes in the order of the subtrees, the same way XyDiff does.

The DOCTREEDIFF [21] approach considers that an XML document is an ordered tree. The algorithm considers all the standard operations in addition to the move operation. The authors introduce a model that considers the neighboring nodes to achieve algorithm efficiency to generate good *deltas*. Their *delta* model allows the reconstruction of the previous version from the newer one. The key idea of the algorithm for detecting differences, matching and, generating the delta is based on the LCS (Longest Common Subsequence) algorithm [64] and hash functions.

Tekli and Chbeir [31] also proposed a tree-based approach for XML grammar matching using edit distances. Their approach uses a tree representation model for the XML grammar, which considers the hierarchical aspect and constraints for XML elements. This hierarchical tree is used for calculating the tree edit distance [65] to determine structural matches. Similarly, they also proposed another approach [33] to consider sub-tree structural similarities when comparing XML documents. This other approach targets only the XML structure and disregards the content, making it useful for applications that query the document structure. Like the previous approach, this one also uses the tree edit distance to capture structure similarities between documents. However, unlike the former approach, they also use information retrieval semantic assessment to capture semantic changes in the structure of the documents. This semantic similarity algorithm uses a weighted semantic network as an input, which acts similarly to a dictionary for defining semantically similar words, to determine similarities between element names. Tekli et al. [34] also proposed the Differential SOAP Multicasting (DSM) approach for improving SOAP protocol for XML documents. DSM identifies common patterns between SOAP messages, using their previously established tree edit distance, to multicast similar parts of the message to minimize network traffic. These approaches mainly focus on comparing XML schema, and not content changes from different versions of the same document, which is complementary to our approach.

The diffi approach [12] compares two files to calculate their differences and return a delta file describing the modifications needed to transform the former file into the later. One interesting aspect of diffi is that it compares multiple levels of abstractions instead of only one level, as is common in most diff tools. This allows for the diffi approach to compare files of different formats, including XML documents. The diffi approach can detect operations such as *addition*, *deletion*, *moving*, *wrapping*, and *splitting*. The diffi approach first decodes both files to identify the abstraction level that they match and then compares the files by computing the deltas for each comparable abstraction level. Lastly, it serializes the produced deltas for each abstraction level to generate the final delta file (or as the authors call it, the “patch file”). However, the authors do not show experimental results nor evaluate the complexity of the proposed algorithm.

The XSDF approach [32] produces a semantic XML tree using lexical knowledge bases to identify semantic relationships. This approach selects ambiguous elements using an ambiguity degree measure for the disambiguation of XML nodes from different documents. The user can customize the disambiguation based on context and/or concept according to her needs. This allows identifying documents that convey the same data but use different tags and structure. SemIndex+ [66] is another graph-based index approach, which maps two textual documents and a semantic knowledge base to create a semantic aware indexing system to provide a general keyword query model with broader semantic coverage. This allows the query to be more semantically flexible. Differently from XChange, which focuses on content changes with the same structure, these approaches focus on document structure with semantically similar tags.

Phoenix [20] calculates the similarity of two versions of a given XML document with the same schema. It tries to match elements from one version to the other. For that, it uses the element name, its content, its attributes, and its children. The matches produced by the similarity calculation are then used to produce the optimum delta, which maximizes the global similarity among elements. This delta is then visually shown to the user. XChange uses Phoenix to match elements from one version to the other before calculating the semantic diff.

Lastly, some approaches focus on **mining changes** in XML documents. The work developed by Rusu *et al.* [22] aims at mining association rules from XML documents. The work focuses on the mining of dynamic XML documents, that is, documents that may suffer changes one their structure or content over time, such as the

registration of employees of a company or products of a supermarket. The approach proposes the construction of a generic algorithm for extracting association rules in XML documents, based on Apriori [50]. The proposal uses X-Diff [27] to generate the consolidated delta with the history of changes that occurred in the versions of the XML document. The extracted rules can detect relations between the changes in different parts of documents – that is, it can detect relationships among the modifications, deletions, or insertions of elements.

Another work in this line deals with the problem of discovering structures that frequently change according to certain patterns, considering their dynamic nature [29]. This approach focuses on mining dynamic structures based on version patterns of unordered versions of XML documents. The authors use a modified version of the X-Diff algorithm [27] in the mining process to support the diff process and propose an algorithm to identify the changing structures.

Our work is a pioneer in the sense that it can detect the meaning of the changes of two versions of an XML document, unlike existing work in the literature.

6. Conclusion

This paper presents XChange, an approach to help to identify the reason behind modifications in XML document versions. The identification process is based on the analysis of the syntactic modifications in attributes and elements of the document. For this, XChange uses an inference mechanism based on Prolog. XChange can perform the analysis of sequential versions that are not necessarily consecutive. XChange works on the identification of corresponding elements in two versions in two ways: using *matching by key* or *matching by similarity*. In the key matching approach, the user must indicate a key attribute. Depending on how the user manages the XML documents, there is no guarantee that the key-attribute value remains the same between versions (for example, a typing error may occur in the value of the key attribute, and later someone fixes that error in a future version). To mitigate this problem, we also provided an approach based on similarity analysis.

Although XChange uses Prolog to represent the document and the inference rules, it does not require the user to be a Prolog expert since it uses an interface that automatically generates Prolog rules from a selection of options at a higher abstraction level. Finally, the generated rules are valid for all documents in the same domain so that the rule configuration step may occur only once for a given domain. Note that the domain expert may revise the rules at any time for a given domain.

Another feature offered by XChange is the semiautomatic construction of semantic enrichment rules based on the mining of elements that were frequently modified together. The Apriori algorithm identifies *itemsets* of elements that are frequently changed together and suggests semantic enrichment rules. Then, the domain expert validates and name these rules. Rules produced by the mining process may also be discarded by the domain expert, as he/she may see fit.

We evaluated our approach through an experimental study aiming at answering two research questions related to the effectiveness and efficiency of users supported by XChange in understanding the evolution of XML documents. We contrasted the use of our tool with the use of the state-of-the-art approach (X-Diff) in producing syntactic diffs. XChange allowed users to be more effective and more efficient when compared with users supported by X-Diff. Participants indicated that the resulting diff file from X-Diff is much smaller than that of XChange. However, the participants had to go through the entire diff document to complete the tasks since X-Diff does not summarize the answers. Therefore, according to the participants, tasks completed with X-Diff was more time consuming, less intuitive, and more error-prone even though the resulting delta was smaller.

Conversely, the summarization offered by XChange makes the resulting delta very large, depending on the original XML document. It also does not show the delta information embedded in the original document – a distinguishing feature of X-Diff. Meanwhile, X-Diff uses the original document to present the diff and shows the complete document. Moreover, the current version of our approach assumes that both XML documents follow the same schema. In situations where documents follow different schemas (or different versions of the same schema), we suggest applying a preprocessing step using XSLT or another equivalent technology to first unify the schemas before running our approach.

As future work, we plan to support other languages, such as JSON, due to its increase in popularity in recent years. We also plan to extend XChange, which considers only the evolution of the data, so that versions with

different schemas can be used in semantic diff. Another possibility would be to allow users to identify XML element tags that are equivalent in similar schemas. That is, schemas that have attributes with different names but the same meaning. Furthermore, we intend to adapt XChange to provide semantic information from merges. Another interesting future work would be adding an ontology to consider semantically similar words when identifying the rules using Apriori.

Acknowledgements

We would like to thank CNPq, FAPERJ, and CAPES for financial support.

References

[1] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation., (2008). http://www.w3.org/TR/xml/ (accessed November 22, 2018).
[2] V. Getov, e-Science: The Added Value for Modern Discovery, Computer. 41 (2008) 30–31.
[3] M. Argüello, J. Des, M.J. Fernandez-Prieto, R. Perez, H. Paniagua, Executing Medical Guidelines on the Web: Towards Next Generation Healthcare, in: T.A.B. MSc, R.E.Bs. MSc, M.P.D. AMBA MBA,, MBCS (Eds.), Appl. Innov. Intell. Syst. XVI, Springer London, 2009: pp. 197–210. http://link.springer.com/chapter/10.1007/978-1-84882-215-3_15 (accessed August 17, 2014).
[4] P.T.T. Thuy, Y.-K. Lee, S. Lee, Semantic and structural similarities between XML Schemas for integration of ubiquitous healthcare data, Pers. Ubiquitous Comput. 17 (2013) 1331–1339. https://doi.org/10.1007/s00779-012-0567-5.
[5] M. Hallo Carrasco, M.M. Martínez-González, P. De La Fuente Redondo, Data Models for Version Management of Legislative Documents, J. Inf. Sci. 39 (2013) 557–572. https://doi.org/10.1177/0165551512473723.
[6] CNPQ, CNPQ, Plataforma Lattes. (2017). http://lattes.cnpq.br/ (accessed August 15, 2012).
[7] Portal Brasileiro de Dados Abertos, Dados Abertos, (2017). http://dados.gov.br/ (accessed January 2, 2017).
[8] Wikimedia, Wikimedia, (2017). https://dumps.wikimedia.org/ (accessed January 2, 2017).
[9] M. Ley, DBLP: Some Lessons Learned, Proc. VLDB Endow. PVLDB. 2 (2009) 1493–1500. https://doi.org/10.14778/1687553.1687577.
[10] M. Ley, DBLP in XML, DBLP. (2003). http://dblp.uni-trier.de/xml/ (accessed November 22, 2018).
[11] R. Al-Ekram, A. Adma, O. Baysal, diffX: An Algorithm to Detect Changes in Multi-version XML Documents, in: Proc. 2005 Conf. Cent. Adv. Stud. Collab. Res., IBM Press, Toranto, Ontario, Canada, 2005: pp. 1–11. http://dl.acm.org/citation.cfm?id=1105634.1105635 (accessed November 18, 2016).
[12] G. Barabucci, Diffi: Diff Improved; a Preview, in: Proc. ACM Symp. Doc. Eng. 2018, ACM, New York, NY, USA, 2018: p. 38:1–38:4. https://doi.org/10.1145/3209280.3229084.
[13] S.S. Chawathe, H. Garcia-Molina, Meaningful Change Detection in Structured Data, in: ACM SIGMOD Int. Conf. Manag. Data, ACM, New York, USA, 1997: pp. 26–37. https://doi.org/10.1145/253260.253266.
[14] G. Cobena, S. Abiteboul, A. Marian, Detecting changes in XML documents, in: Int. Conf. Data Eng. ICDE, IEEE Computer Society, San Jose, California, USA, 2002: pp. 41–52. https://doi.org/10.1109/ICDE.2002.994696.
[15] J. Jacob, A. Sachde, S. Chakravarthy, CX-DIFF: A Change Detection Algorithm for XML Content and Change Presentation Issues for WebVigiL, in: Concept. Model. Nov. Appl. Domains, Jeusfeld, ManfredA. and Pastor, Óscar, 2003: pp. 273–284.
[16] J. Jacob, A. Sachde, S. Chakravarthy, CX-DIFF: A Change Detection Algorithm for XML Content and Change Visualization for WebVigiL, Data Knowl. Eng. 52 (2005) 209–230. https://doi.org/10.1016/j.datak.2004.05.006.

[17] S. Lim, Y.-K. Ng, An Automated Change Detection Algorithm for HTML Documents Based on Semantic Hierarchies, in: Int. Conf. Data Eng. ICDE, IEEE Computer Society, Heidelberg, Germany, 2001: pp. 303–312.
[18] T. Lindholm, A Three-way Merge for XML Documents, in: ACM Symp. Doc. Eng. DocEng, ACM, Milwaukee, Wisconsin, USA, 2004: pp. 1–10. https://doi.org/10.1145/1030397.1030399.
[19] A. Marian, S. Abiteboul, G. Cobena, L. Mignet, Change-centric management of versions in an XML warehouse, in: Int. Conf. Very Large Data Bases VLDB, Morgan Kaufmann Publishers Inc., Roma, Italy, 2001: pp. 581–590.
[20] A. Oliveira, G. Tessarolli, G. Ghiotto, B. Pinto, F. Campello, M. Marques, C. Oliveira, I. Rodrigues, M. Kalinowski, U. Souza, L. Murta, V. Braganholo, An efficient similarity-based approach for comparing XML documents, Inf. Syst. 78 (2018) 40–57. https://doi.org/10.1016/j.is.2018.07.001.
[21] S. Rönnau, G. Philipp, U.M. Borghoff, Efficient Change Control of XML Documents, in: ACM Symp. Doc. Eng. DocEng, ACM, Munich, Germany, 2009: pp. 3–12. https://doi.org/10.1145/1600193.1600197.
[22] L.I. Rusu, W. Rahayu, D. Taniar, Mining Changes from Versions of Dynamic XML Documents, in: Int. Conf. Knowl. Discov. XML Doc. KDXD, Springer-Verlag, Berlin, Heidelberg, 2006: pp. 3–12. https://doi.org/10.1007/11730262_3.
[23] R.C. Santos, C.S. Hara, A Semantical Change Detection Algorithm for XML, in: Softw. Eng. Knowl. Eng. SEKE, Knowledge Systems Institute Graduate School, Boston, Massachusetts, USA, 2007: pp. 438–443.
[24] Y. Song, S.S. Bhowmick, C.F. Dewey, Jr., BioDIFF: an effective fast change detection algorithm for biological annotations, in: Int. Conf. Database Syst. Adv. Appl. DASFAA, Springer-Verlag, Berlin, Heidelberg, 2007: pp. 275–287.
[25] S. Sundaram, S.K. Madria, A change detection system for unordered XML data using a relational model, Data Knowl. Eng. 72 (2012) 257–284. https://doi.org/10.1016/j.datak.2011.11.003.
[26] C. Thao, E.V. Munson, Using Versioned Tree Data Structure, Change Detection and Node Identity for Three-way XML Merging, in: ACM Symp. Doc. Eng. DocEng, ACM, Manchester, UK, 2010: pp. 77–86. https://doi.org/10.1145/1860559.1860578.
[27] Y. Wang, D.J. DeWitt, J.-Y. Cai, X-Diff: an effective change detection algorithm for XML documents, in: Int. Conf. Data Eng. ICDE, IEEE Computer Society, Bangalore, India, 2003: pp. 519–530. https://doi.org/10.1109/ICDE.2003.1260818.
[28] H. Xu, Q. Wu, H. Wang, G. Yang, Y. Jia, KF-Diff+: Highly Efficient Change Detection Algorithm for XML Documents, in: R. Meersman, Z. Tari (Eds.), Move Meaningful Internet Syst. 2002 CoopIS DOA ODBASE, Springer Berlin Heidelberg, 2002: pp. 1273–1286. https://doi.org/10.1007/3-540-36124-3_80.
[29] Q. Zhao, S.S. Bhowmick, S. Madria, Discovering Pattern-Based Dynamic Structures from Versions of Unordered XML Documents, in: Data Warehous. Knowl. Discov., Springer Berlin Heidelberg, Zaragoza, Spain, 2004: pp. 77–86.
[30] C.F. Dorneles, M.F. Nunes, C.A. Heuser, V.P. Moreira, A.S. da Silva, (Edleno S. de) Moura, A strategy for allowing meaningful and comparable scores in approximate matching, Inf. Syst. 34 (2009) 673–689. https://doi.org/10.1016/j.is.2009.05.002.
[31] J. Tekli, R. Chbeir, Minimizing user effort in XML grammar matching, Inf. Sci. 210 (2012) 1–40. https://doi.org/10.1016/j.ins.2012.04.026.
[32] J. Tekli, N. Charbel, R. Chbeir, Building semantic trees from XML documents, J. Web Semant. 37–38 (2016) 1–24. https://doi.org/10.1016/j.websem.2016.03.002.
[33] J. Tekli, R. Chbeir, A novel XML document structure comparison framework based-on sub-tree commonalities and label semantics, J. Web Semant. 11 (2012) 14–40. https://doi.org/10.1016/j.websem.2011.10.002.
[34] J. Tekli, E. Damiani, R. Chbeir, Using XML-Based Multicasting to Improve Web Service Scalability, Int. J. Web Serv. Res. 9 (2012) 1–29. https://doi.org/10.4018/jwsr.2012010101.
[35] P. Buneman, S. Davidson, W. Fan, C. Hara, W.-C. Tan, Keys for XML, Comput. Netw. 39 (2002) 473–487. https://doi.org/10.1016/S1389-1286(02)00223-2.
[36] P. Buneman, S. Davidson, W. Fan, C. Hara, W.-C. Tan, Reasoning about keys for XML, Inf. Syst. 28 (2003) 1037–1063. https://doi.org/10.1016/S0306-4379(03)00028-0.
[37] D.C. Fallside, P. Walmsley, XML Schema Part 0: Primer Second Edition. W3C Recommendation., (2004). http://www.w3.org/TR/xmlschema-0/ (accessed March 18, 2015).

[38] M.Y. Maarouf, S.M. Chung, XML Integrated Environment for Service-Oriented Data Management, in: 2008 20th IEEE Int. Conf. Tools Artif. Intell., 2008: pp. 361–368. https://doi.org/10.1109/ICTAI.2008.150.
[39] S. Grijzenhout, M. Marx, The quality of the XML Web, Web Semant. Sci. Serv. Agents World Wide Web. 19 (2013) 59–68. https://doi.org/10.1016/j.websem.2012.12.001.
[40] J. Vyhnanovská, I. Mlýnková, Interactive inference of XML schemas, in: 2010 Fourth Int. Conf. Res. Chall. Inf. Sci. RCIS, 2010: pp. 191–202. https://doi.org/10.1109/RCIS.2010.5507523.
[41] T. Grabs, H.-J. Schek, PowerDB-XML: A Platform for Data–Centric and Document–Centric XML Processing, in: Z. Bellahsène, A.B. Chaudhri, E. Rahm, M. Rys, R. Unland (Eds.), Database XML Technol., Springer, Berlin, Heidelberg, 2003: pp. 100–117. https://doi.org/10.1007/978-3-540-39429-7_7.
[42] Mayor's Office of Information Technology, OpenBaltimore, (2016). https://data.baltimorecity.gov/ (accessed October 14, 2016).
[43] F. Campello, B. Pinto, G. Tessarolli, A. Oliveira, C. Oliveira, M.O. Junior, L. Murta, V. Braganholo, A similarity-based approach to match elements across versions of XML documents, in: Simpósio Bras. Banco Dados SBBD, SBC, Curitiba, PR, Brasil, 2014.
[44] D. Lima, C. Delgado, L. Murta, V. Braganholo, Towards Querying Implicit Knowledge in XML Documents., J. Inf. Data Manag. JIDM. 3 (2012) 51–60.
[45] S.S. Huang, T.J. Green, B.T. Loo, Datalog and Emerging Applications: An Interactive Tutorial, in: Int. Conf. Manag. Data SIGMOD, ACM, New York, NY, USA, 2011: pp. 1213–1216. https://doi.org/10.1145/1989323.1989456.
[46] S.S. Huang, T.J. Green, B.T. Loo, Datalog and emerging applications: An interactive tutorial, in: SIGMOD, 2011.
[47] H. Boley, The rule markup language: RDF-XML data model, XML schema hierarchy, and XSL transformations, in: Int. Conf. Web Knowl. Manag. Decis. Support, Springer-Verlag, Berlin, Heidelberg, 2003: pp. 5–22. http://dl.acm.org/citation.cfm?id=1767370.1767373 (accessed September 6, 2012).
[48] I. Bratko, Prolog programming for artificial intelligence, Addison Wesley, Harlow, England; New York, 2001.
[49] Santos, Prolog Versus XQuery Processors: A Performance Evaluation Of XML Queries Processing Methods, Dissertação (Mestrado em Computação), Universidade Federal Fluminense, 2015.
[50] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Int. Conf. Very Large Data Bases VLDB, Jorge B. Bocca and Matthias Jarke and Carlo Zaniolo, Santiago de Chile, Chile, 1994: pp. 487–499.
[51] R. Agrawal, T. Imieliński, A. Swami, Mining Association Rules Between Sets of Items in Large Databases, in: Proc. 1993 ACM SIGMOD Int. Conf. Manag. Data, ACM, New York, NY, USA, 1993: pp. 207–216. https://doi.org/10.1145/170035.170072.
[52] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA Data Mining Software: An Update, ACM SIGKDD Explor. Newsl. 11 (2009) 10–18. https://doi.org/10.1145/1656274.1656278.
[53] R. Elmasri, S. Navathe, Fundamentals of Database Systems, 6th ed., Addison-Wesley, 2010.
[54] N. Juristo, A.M. Moreno, Basics of Software Engineering Experimentation, Kluwer Academic Publishers, 2001.
[55] W.J. Conover, Practical Nonparametric Statistics, 3rd ed., Wiley, New York, NY, USA, 1999.
[56] P. Royston, Remark {AS R94}: A Remark on Algorithm {AS 181}: The {W}-test for Normality, J. R. Stat. Soc. Ser. C Appl. Stat. 44 (1995) 547–551. https://doi.org/10.2307/2986146.
[57] F. Wilcoxon, Individual Comparisons by Ranking Methods, Biom. Bull. 1 (1945) 80–83. https://doi.org/10.2307/3001968.
[58] K. Strassburger, F. Bretz, Compatible simultaneous lower confidence bounds for the Holm procedure and other Bonferroni-based closed tests, Stat. Med. 27 (2008) 4914–4927. https://doi.org/10.1002/sim.3338.
[59] N. Cliff, Ordinal Methods for Behavioral Data Analysis, Psychology Press, Mahwah, N.J, 1996.
[60] J. Romano, J.D. Kromrey, J. Coraggio, J. Skowronek, Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys, in: Annu. Meet. Fla. Assoc. Institutional Res., 2006: pp. 1–33.
[61] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[62] S. Chamakura, A. Sachde, S. Chakravarthy, A. Arora, WebVigiL: Monitoring Multiple Web Pages and Presentation of XML Pages, in: Int. Conf. Data Eng. ICDE - Workshop, IEEE Computer Society, Tokyo, Japan, 2005: p. 1276. https://doi.org/10.1109/ICDE.2005.306.
[63] J. Tekli, An Overview on XML Semantic Disambiguation from Unstructured Text to Semi-Structured Data: Background, Applications, and Ongoing Challenges, IEEE Trans. Knowl. Data Eng. 28 (2016) 1383–1407. https://doi.org/10.1109/TKDE.2016.2525768.
[64] D. Maier, The Complexity of Some Problems on Subsequences and Supersequences, J ACM. 25 (1978) 322–336. https://doi.org/10.1145/322063.322075.
[65] K. Zhang, D. Shasha, Simple Fast Algorithms for the Editing Distance between Trees and Related Problems, SIAM J. Comput. 18 (1989) 1245–1262. https://doi.org/10.1137/0218082.
[66] J. Tekli, R. Chbeir, A.J.M. Traina, C. Traina, SemIndex+: A semantic indexing scheme for structured, unstructured, and partly structured data, Knowl.-Based Syst. 164 (2019) 378–403. https://doi.org/10.1016/j.knosys.2018.11.010.