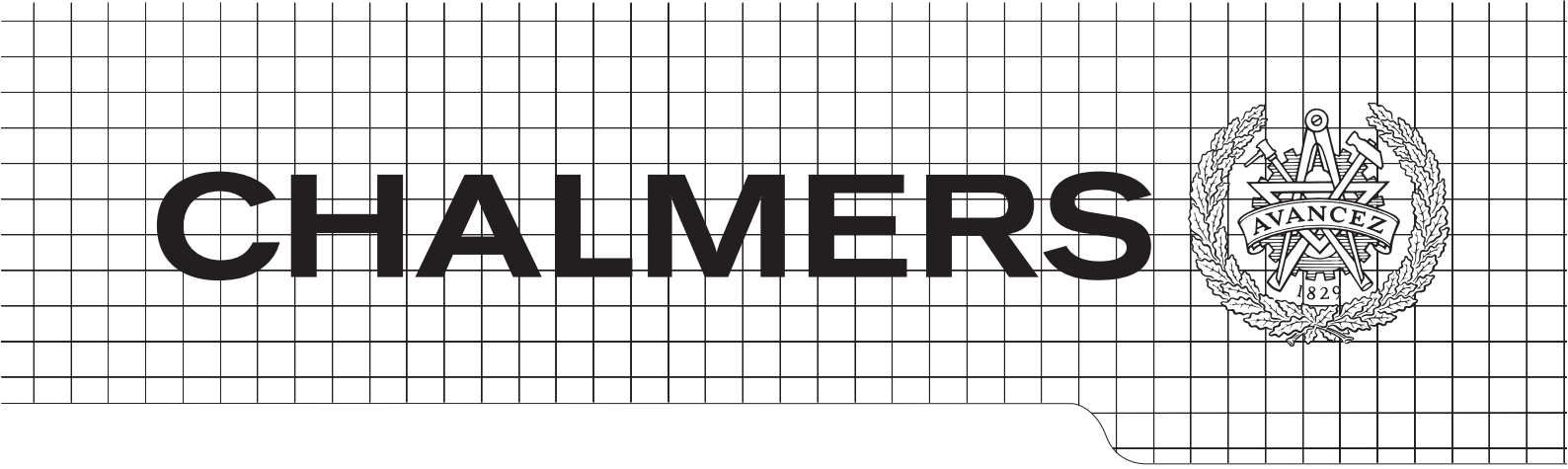


Version Control of structured data:
A study on different approaches in XML

Master Thesis in Software Engineering

ERIK AXELSSON
SÉRGIO BATISTA

Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Abstract

The structured authoring environment has been changing towards a decentralised form of authoring. Existing version control systems do not handle these documents adequately, making it very difficult to have parallel authoring in a structured environment. This study attempts to find better alternatives to the existing paradigms and tools for versioning XML documents.

To achieve this, the DESMET methodology for evaluating software engineering methods and tools was applied, with both a Feature Analysis and a Benchmark Analysis being performed.

Concerning the feature analysis, the results demonstrate that the XML-aware tools are, as expected, better at XML specific concerns, such as considering the history of a specific node. Conversely, the non-XML-aware ones are not able to achieve good results in the XML specific concerns, but do achieve a high score when considering project maturity or general repository management features. Regarding performance, this study concludes that XML-aware tools bring a considerable overhead when compared to the non-XML-aware tools.

This study concludes that the selection of an approach to versioning XML should be dependent of the priorities of the documentation project.

Keywords: *Structured documentation, XML, Version Control, distributed collaboration, Git, Sirix, XChronicler, temporal databases, DESMET.*

Acknowledgements

The authors would like to thank both the academic supervisor Morgan Ericsson and examiner Matthias Tichy for their feedback and support throughout this thesis work. We would also like to thank the supporting company – Simonsoft – for giving us the opportunity to conduct this research on their premises. We thank our industry supervisors Staffan Olsson and Thomas Åkesson for their continuous support throughout our work. Special thanks to our friends and family.

Erik and Sérgio, Gothenburg - June 2015

Contents

List of Tables

List of Figures

Listings

1

Introduction

XML documentation production is moving towards decentralised production and faster release cycle. Documentation authors have the need to manage different documents and versions of them. It is also common for a set of these documents to share similar content. Eventually, when part of the content is changed, that specific change might be required to be reflected in some other documents. With the decentralised authoring environment and having many document maintainers, there is the need to keep eXtensible Markup Language (XML) documentation version controlled with all the complexity that it involves. All of this should be simple to achieve and effortless.

Considering that background, there is the need for new tools to help authors keeping up with the rapid development pace while maintaining quality.

In this study we design, implement, compare and evaluate different implementations of different paradigms to XML version control. Based on the outcome of the comparison, we suggest the best fitting approaches regarding this problem.

1.1 Purpose of the study

The purpose of this study is to investigate, evaluate and suggest best-fit approaches to perform versioning in structured documents in the form of ordered XML. Rönnau [2] states that "most supporting tools like version control systems, or business process systems do not handle these [XML] documents adequately. Parallel editing of documents across network and system borders is almost impossible". This study intends to provide maintainers of multiple XML documents with a better approach to XML versioning.

This research can be contained within the Software Engineering Knowledge Area of Software Configuration Management [3].

1.2 Statement of the problem

With the fast pace in releasing new and more complex products, as well as the need for maintaining older ones, the authoring of structured documentation, specifically product manuals, is moving from a centralised authoring paradigm towards a decentralised one. This paradigm shift is happening in order to increase the rate and easiness of production as well as increasing the quality of these same documents. To achieve this, more specialised contributors are deemed necessary. These are mostly untrained authors, that don't want to dwell with complex documentation processes. Therefore, to get their contribution, the complexity of the authoring process needs to decrease.

1.2.1 So what is this complexity about?

The complexity derives from having many different versions of the same documents along with other similar ones that may share parts among them.

A solution to minimise the shared parts issue currently in use is referred as modularisation. Modularisation is the manual insertion of cross-references to other documents that share the same content, this part will then be imported when compiling the document for release. The problem with this manual approach is that it requires design upfront in order to select and extract which parts are going to be reused later or some refactoring effort later on. Instead, the more common solution in use is the plain copy-paste text approach because it is easier and faster to do. There is a trade-off with this approach, by saving time during the creation of the document, one highly increases its maintenance complexity.

Tree based data structures have different characteristics from unstructured text ones, with the latter having well established and efficient ways of versioning. Systems such as Concurrent Version System (CVS), Subversion (SVN), Git or mercurial, are today, for example, an essential part of any software codebase. For the tree-based ones though, there are no currently well established ways of versioning this type of data in an efficient way.

1.3 Research questions

This thesis has a main research goal that is to understand what is a good approach to the versioning of an XML document. As mentioned before, the current state of the industry is to use non-XML-aware tools, relying on currently well established version control systems that do not take into account the specificities of a tree-based document.

1.3.1 RQ0: What is a good approach to versioning an XML document?

The main research question that this work tries to answer is which are the best approaches to version XML, and under which circumstances those approaches should be used. In order to answer this, there is the need to answer the following sub-questions:

RQ1: What features are required for a version control system to have in a documentation context?

In order to understand the document maintainers' needs from the version control system, there is the need to elicit and specify the requirements.

RQ2: Of the previously identified features, which ones do XML-aware and non-XML-aware tools have?

In order to learn what are the most appropriate tools for versioning XML, there is the need to understand which tools contain which of the required features.

RQ3: How much overhead does an XML-aware tool carry when compared to a non-XML-aware one?

In order for a complete comparison, a cost/benefit analysis is also required, therefor we need to understand how expensive the XML-aware tools are when compared to non-XML-aware ones.

1.4 Scope and Delimitations

This study focuses on XML documents and their versioning. There are numerous version control system available that, due to the limited time-frame of this research, can not be taken into account. As such, as an initial delimitation, this research focuses only on Free and Open-Source Software (FOSS) tools, as a way to limit the amount of tools as well as free the research from the constraints of restrictive licenses and the most likely need to study the source code of the tool in question.

Although the overall end goal is to tackle a production system, for this research scenario the focus excludes the repository structure, this includes branching/forking of repositories and similar features. The assumption being that the benefit of addressing such scenarios does not out the research costs, mainly in what time is concerned.

In what regards the XML structure, as it contains many intricacies, some parts of the specification have to be left out. The rationale is again that the cost/benefit of addressing them would not be deemed enough.

- Only ordered XML, i.e. the order of nodes matters (but not the order of attributes);
- Only XML for documentation purposes, any other usage for XML is considered out of the scope of this study.
- Within structured documentation, this study will not deal with Open Document Format (ODF) or similar. It will focus on "pure" XML, this means Comments, Processing Instructions, document type definition, Schemas and similar will not take part on the scope of this research.

1.5 Outline of the Report

This thesis report is divided into eight chapters – This first chapter introduces and provides an overview of the study, its motivations and goals. Chapter number two provides the required background for the remainder of the report. In chapter number three we discuss related work and what has been done before this work and how it relates to the topic. The fourth chapter presents the method followed by this research work including the data collection approach for each research question. In chapter five, we discuss the implementation part of the evaluation, including the overall architecture of the system, and more specifically how some of the tests were performed. The sixth chapter presents the results of the research work and the seventh chapter presents the discussion of those results along with the identified validity threats to this study. The eight and final chapter presents the conclusions and possible future work in this field.

In the appendices we include the elicited User Stories (appendix A on page 72); the Feature List used for the evaluation with their metrics (appendix B on page 75); the detailed Benchmark Results (appendix C on page 81); the data used for the analysis of the projects behind the two final products evaluated (appendix D on page 83); the script used for the pre-processing steps of the benchmark analysis (appendix E on page 84); the specification of the environment used for the benchmark analysis (appendix F on page 86); and finally, the source code of an example test script used on the evaluation (appendix G on page 87).

2

Foundations

This chapter lays out the foundational background for this thesis work. It addresses 4 main areas: XML, XQuery, XML Storage, and XML Version Control. It is assumed that the reader has a basic knowledge within each of the areas and this section intends to set a common baseline of understanding that will be required for the next chapters.

2.1 XML

XML [4, 5] is a markup language with its specification being a World Wide Web Consortium (W3C) recommendation since 1998 [4]. It is used to represent structured information in documents, data, configuration, books, transactions, invoices, and much more.

It is designed to be read and understood by both machines and humans, with its main goal to transport and store data and focusing on what data is rather than on what data looks, figure 2.1 on the next page contains a basic example of an XML structure.

2.1.1 Key Constructs

This section presents a list of the most common XML constructs, it is by no means exhaustive and intends only to put the reader up to speed with some of the terms mentioned in this report.

Markup and Content

The set of characters in an XML document are divided into markup and content. In a general sense, strings that are delimited by angle-brackets (< and >) or by an ampersand and a semi-colon (`&` and `;`) constitute the markup and the remaining strings are

```
<foo attr="norf">
    <bar>
        baz
    </bar>
    <qux />
</foo>
```

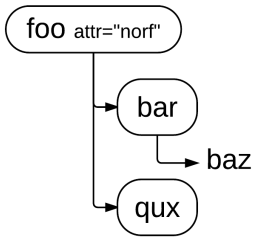


Figure 2.1: XML structure and its equivalent tree representation

considered content. *N.B.* There are some exceptions to these rules, but these are out of the scope of this study.

Tag

An XML tag is a markup construct delimited by angle-brackets (< and >). This construct might be an opening- (`<tag>`), closing- (`</tag>`) or self-closing-tag (`<tag/>`).

Element

An element is a component that starts with an opening-tag and ends with the correspondent closing-tag (it can also be contained within a self-closing-tag). Its content, if existent, are the characters between the delimiting tags (opening and closing).

The element content can also contain markup that may be other elements, being those the child elements to this one.

Attribute

An XML attribute is often used to describe the data rather than containing relevant data itself[6]. It is added within the tag of an element and has a key and a value. In between both, the equals sign is used (`=`) and the value is always quoted, either by a single quote (`'`) or by a double one (`"`). An example of a tag element with the attribute *id* is as follows: `<tag id="5" />`.

XML Declaration

The XML Declaration is an pseudo-element that can be present on the first line of a document. It consists of the xml `version` and the text `encoding`. It can also include the "pseudo-attribute"[7] `standalone` which essentially notifies the parser if the document has external documents to be fetched in order to be well-formed.

Example: `<?xml version="1.0" encoding="UTF-16" standalone="no" ?>`

2.1.2 Well-formedness

Even though XML is a meta-markup language, in the sense that one can come up with the tags and attributes as one writes the document, the document still needs to follow some ground rules (more than 100 different ones). This is due to the specification [5] strictly forbidding an XML Parser from fixing or even understand malformed documents, unlike, for example, HTML. Well-formedness is then the lowest level required for the XML to be parsed.

The complete set of rules can be found in the XML specification[5].

2.1.3 XML Canonicalisation

Canonical XML is a separate W3C Recommendation[8, 9] that converts an XML document into a "single stand-alone file that can be compared byte-for-byte with other canonical XML Documents."[7]

The XML Canonicalization (C14N) process requires a set of 15 steps that transform the document after which, and according to the W3C[5], "if two XML documents have the same canonical form, then the two documents are logically equivalent within the given application context".

Below, we show some examples of the normalisation steps mentioned above:

- Normalisation of white-spaces within an element;
- Conversion of self-closing elements to ones with opening and closing tag, e.g. `<foo />` becomes `<foo></foo>`;
- Sorting of attributes within an element according to their unicode character number, e.g. `<foo c="" b="" a="" ></foo>` becomes `<foo a="" b="" c="" ></foo>`;
- UTF-8 is used for encoding;
- Removal of superfluous namespace declarations;

2.1.4 XML in Structured Documentation

As described by the W3C XHTML2 working group: "XML is the universal format for structured documents and data on the Web."[10] Along with other formats, such as SGML, LaTeX, XML allows the separation of content from visualisation and allows for the validation of content through the uses of schemas.

Note that in our specific situation XML for documentation purposes the order of the nodes matter as opposed to regular XML.

2.2 XQuery

XQuery is a query language for XML, in many ways is similar to Standard Query Language (SQL). It is a W3C recommendation since 2007 [11] and with its latest version (3.0) released while this thesis work was undergoing in 2014 [12]. The following pages describe what the authors consider to be important aspects of the language that were used in this research work.

2.2.1 FLWOR expressions

FLWOR (pronounced 'flower') is a type of XQuery expressions, similar to a regular SQL expression. It stands for `for`, `let`, `where`, `order by`, `return`. Listing 2.1 displays an example extracted from [11, 3.8 FLWOR Expressions] where the query can be loosely translated into "Get all departments that have more than ten employees, order these departments by decreasing average salary, and return a report of department numbers, head counts and average salary in each big department" [13].

Listing 2.1: XQuery FLWOR example

```
for $d in fn:doc("depts.xml")/depts/deptno
let $e := fn:doc("emps.xml")/emps/emp[deptno = $d]
where fn:count($e) >= 10
order by fn:avg($e/salary) descending
return
    <big-dept>
      {
      $d,
      <headcount>{fn:count($e)}</headcount>,
      <avgsal>{fn:avg($e/salary)}</avgsal>
      }
    </big-dept>
```

In XQuery 1.0, the `for` and `let` statements can be multiple, are interchangeable and can only eXist before the `where` clause. The most recent version (XQuery 3.0 [12]) allows both statements to be included also after the `where` clause, allowing for the simplification of many existing nested queries. It also extends the FLWOR expressions by adding newer clauses (`group by`, `count`, and `window`, among other changes (none covered by the scope of this work).

2.2.2 XPath

XPath is a query language mainly used to select/address parts of an XML document. It came out as a W3C recommendation in 1999 with the purpose of being used along with XSLT and with XPointer [14], its use has since then been more differentiated.

Besides being used to address parts of a document, XPath contains numerous standard built-in functions that span from strings/numerical values to date/time manipulations. These functions will not be described here in this report, refer to the W3C recommendation[14] for more information.

Nodes and their relationship

XPath treats each part of an XML document as a `node`. There are seven different types of nodes: `element`, `attribute`, `text`, `namespace`, `processing-instruction`, `comment`, and the `document` itself.

Together with nodes, there are *Atomic Values* — the content of a node with no children or parent.

Finally, XPath categorises the relationships between each node in five different ways:

Parent each node of the type `attribute` or `element` has one and only one parent, at the exception of the `root` element node which has none

Child each `element` node can have zero or more child nodes.

Sibling a node that shares the same parent

Ancestor a node's parent, parent's parent, etc.

Descendant a node's children, children's children, etc.

Axes

In XPath specification there are thirteen different axis that represent the different possible relations among nodes. figure 2.2 on the next page represents visually these axes. These axes can then be used within a location path to target a specific node.

Location Path Expression

Location Path Expressions(LPEs) are used to target selected node(s), they can be either absolute or relative.

In order for the LPE to be absolute, it must start with a forward slash (`/`), otherwise it is considered a relative one.

An LPE has one or more steps, each of these separated by a forward slash `/`. Each of these steps is then evaluated against the nodes in the current node-set.

The syntax for a location step is `axisname::nodetest[predicate]`, where:

axisname defines the tree-relationship between the selected nodes and the current node

nodetest identifies a node within an axis

predicate to further refine the selected node-set (can be zero or more)

2.2.3 XQuery Update Facility

XQuery Update Facility (XQUF) is a W3C recommendation since 2011[15] that extends the XQuery recommendation[11] allowing persistent changes to XQuery and XPath Data model instances.

Expressions

XQUF adds five new kinds of expressions (or update primitives) and related Update Operations[1] to XQuery, these are:

[1]Note that `upd:put` is also an update operation but is out of the scope of this work.

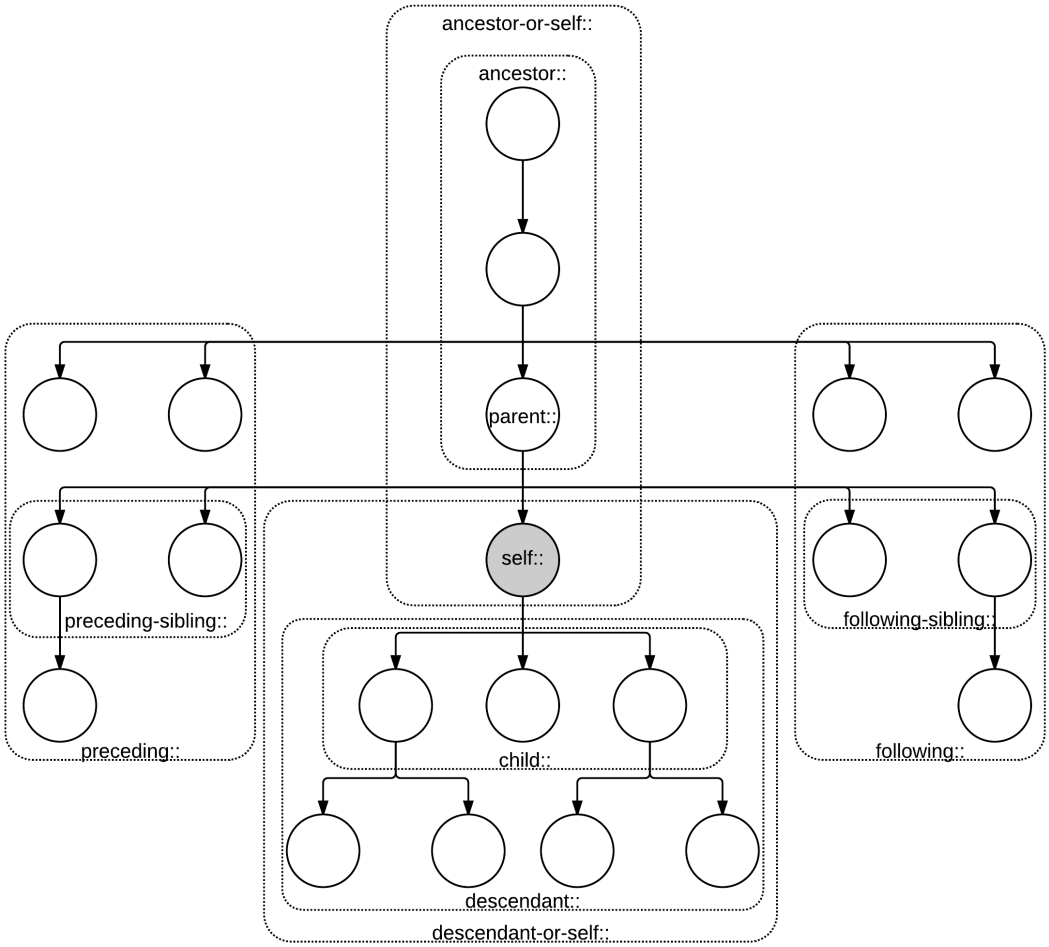


Figure 2.2: XPath Axes

insert a node (or more) inside/before/after a specific node, the related Update Operations (UO) are: `upd:insertBefore`, `upd:insertAfter`, `upd:insertInto`, `upd:insertIntoAsFirst`, `upd:insertIntoAsLast`, `upd:insertAttributes`;

delete one (or more) nodes, UO: `upd:delete`;

replace a node and all of its descendants with another sequence, UO: `upd:replaceNode`, `upd:replaceValue`, `upd:replaceElementContent`;

rename the name property of a node without affecting its contents, UO: `upd:rename`;

transform a node by creating a copy of the original node and modifying its contents.

The standard also specifies 6 update routines. The specifics about each operation and routines are not within the scope of this work, refer to the specification[15] for more details.

Listing 2.2: XQuery Update Facility example

```
1 for $idattr  in doc("data.xml")//ITEM/@Id(: selection :)
2 return (delete  node $idattr ,(: update 1 :)
3 insert  node <NID> {string($idattr)} </NID> (: update 2 :)
4 as first  into $idattr/
```

Pending Update Lists

As defined in the specification, a Pending Update List is an 'unordered collection of update primitives, which represent node state changes that have not yet been applied.' [15]

Within the Pending Update List, the update statements are not executed immediately, but instead collected as update primitives and only applied at the end of the query that contains them.

These primitives are not applied in the order of insertion into the list, instead they are grouped by operation type and then applied in a specific order. A possible order (non-normative) by which these operations are applied is presented in the recommendation [15].

2.3 XML Storage

XML documentation is usually stored in two diverse ways, either within a textual storage system (file system, relational database, among others), or within an XML-aware storage system.

2.3.1 File System

Regarding the first option, arguably all of the external software applications that deal with XML are developed with this scenario in mind and there is no need to adapt them in order for them to access the content. This ease of access also creates an issue with the validation and correctness of the data.

2.3.2 XML Databases

The second option, besides the storage function, and assuming that an XQuery processor is included, allows querying the content. An example of an advantage over the file system is that the results of repetitive queries can be indexed for increased speed. Another upside is that many of the XML-aware tools verify the input, ensuring correctness of syntax.

Bourret[16] divides the XML Databases into two different categories, those that are XML-enabled and those that have a Native XML support.

XML-enabled

XML-enabled are able to interpret XML but does not store it as such. Examples of popular XML-enabled databases are IBM DB2[17], Microsoft SQL Server[18], Oracle Database[19], and PostgreSQL[20], among others.

Native XML

Native XML databases are not only able to interpret the XML, but also stores it as XML. Examples of popular Native XML supported databases are BaseX[21], eXist[22], MarkLogic Server[23], Sedna[24], and Sirix[25]

2.4 XML Version Control

This section explains the current state of the art with regards to version control of XML by highlighting the differences between plain text versioning and tree structures. It is divided into the different features required for the version control: the versioning of different states of the document lifetime, the differencing process between versions, and how merging can be done.

2.4.1 Differencing Plain Text and Tree Structures

When attempting to highlight the differences in plain text, one usually uses a predefined differencing algorithm (e.g. Meyers[26]) to figure out the minimum number of edit operations between two texts and then transpose these operations into an edit script, commonly referred to as diff.

Ideally, this scenario of producing diffs works as in listing 2.3, where the original content and the edit operations combined will always produce the modified text.

Listing 2.3: An ideal scenario for a diff

```
1 diff (original_text, modified_text) --> edit_operations
2 patch(original_text, edit_operations) --> modified_text
```

However, when it concerns tree structures, these linear approaches don't work as efficiently [27]. As the data is contained within structure blocks, the context that represents the change of a content of a single block should reflect a change in the whole block and not only on the line(s) that the change affects. Also, as explained in section 2.1 on page 5 there is no requirement to keep the structure in different lines, in fact, for transferring data across networks, the documents are many times minified, with all of its

content reduced to a few lines, making it very hard to detect what has changed within a line.

A possible way to address this issue is to highlight the changes on a per sub-tree basis with the context being the relation to the neighbouring siblings, parent, and child nodes through the means of a delta[28]. An alternative to this method is to use context fingerprints[29], these take into account the neighbouring nodes within a specific radius while taking into account the document order.

2.4.2 Merging/Patching Documents

Traditionally, there are two different paradigms of merging documents. The first, is when two documents are combined. In order to perform this type of merge, the differences between the two documents are highlighted, and a file containing the differences is generated from it.

The second, used in revision control system, is when there is a common ancestor to the documents being merged – 3-way merge – this approach looks for sections of documents that are common in two of the three files. In the end, two versions of each changed section should exist, with the original (the ancestor) being discarded, and a fourth one being produced, containing the changes existing in both documents.

2.4.3 Versioning

Versioning, also known as revision control and source control, is the act of storing all the versions of a document. This is a very important thing for backup, history and logging who did what and when.

There are different approaches to versioning. Originally, and still quite used, is the storing of different documents with an incremental revision number (or timestamp) in the file name. After, initial version control systems appeared allowing this process to be semi-automatic while adding metadata to each stored revision (e.g. commit message, author, and timestamp) The later tools, contain more advanced features, such as branches, allowing for merging of different versions, the ability of locking files for editing, etc..

2.4.4 The problem of versioning XML using linear approaches

A linear approach to version control fails clearly in three simple scenarios: When using different indentation specifications, when rearranging the tree (e.g. insertion of a parent node), and on changing of the order of attributes within an element.

Regarding the first scenario, a linear based differencing does not understand the difference between significant and non-significant white space. If, for example a document is edited in different tools, each with different indentation specifications (e.g. one uses 2 consecutive white spaces and the other uses 4), the documents might be equivalent but will be recognised as two entirely different ones.

In the second scenario a linear based would only recognise the insertion of the opening and closing element tags, but not that the content structure had clearly changed (we

disregard possible indentation here).

The last scenario presents us with another false positive situation, as the linear approach would detect as a change had occurred, when actually the order of attributes is irrelevant within the context of XML.

3

Related Work

In distributed, cross-organisation authoring environments, a document's life-cycle doesn't follow a linear evolution but instead a collaborative approach [2, 27]. This means that a document is not only being developed iteratively but it might also have different people working on it at the same time. This collaborative Strategy requires more and better merging abilities for which conventional versioning systems are not prepared for with regards to structured data. This issue is recognised by many and with no established nor standardised solution, which leads to many different approaches on how to generate delta files, perform merges or even store the documentation.

3.1 Temporal XML

Chien et al [30] propose efficient storage schemas based on structured change representations and durable node numbers. Wang and Zaniolo built on that and present "efficient techniques to manage multi-version document history" [31] while supporting temporal queries. They do this through the creation of a version-document or V-Document that contains the history of all the nodes that have ever existed in the original document, while using XML queries [11] to retrieve the temporal evolution of the document. Fourny et al [32] use these concepts and elaborate on them with their previous work [33] by the use and extension of standardised XML tools to extract update lists and serialise them in order to query them afterwards. Fourny et al [32] also introduce another approach on versioning XML by the use of node and tree time-lines along with versions creating what they refer to as pi-nodes, pi-trees and pi-forests.

3.1.1 XChronicler and V-Documents

XChronicler is a tool that given many versions of an XML document, it generates another that describes its revision history [31], referred to as V-Document. This document contains all the nodes that have ever existed in the original document, and each has an

identifier to the initial version when was created (`vstart`) and another to when it ceased to exist in its current form (`vend`), this allows for many possible temporal queries. As the most common and basic example, in order to retrieve a specific version of the document, only the nodes for which the `vstart` and `vend` compose the outer boundaries of the version are to be retrieved, i.e. no node that has yet to be created nor one that has ceased to exist are to be displayed. For this thesis work, implementation that was used in this evaluation is the one implemented by Svallfors[34].

Listing 3.1: Example of a V-Document

```
<foo vstart=0 vend="NOW">
    <attr isAttr="yes" vstart=0 vend="NOW">attr</attr>
    <bar vstart=0 vend=1/>
    <baz vstart=1 vend="NOW"/>
</foo>
```

3.1.2 The rise of temporal standards

In [32] Fourny et al propose an extension to the XQuery Data Model with a temporal dimension.

In [35], Lichtenberger proposes adding those axis to treetank and later on, implements them in Sirix[25]. The added commands and their relation are in figure 3.1.

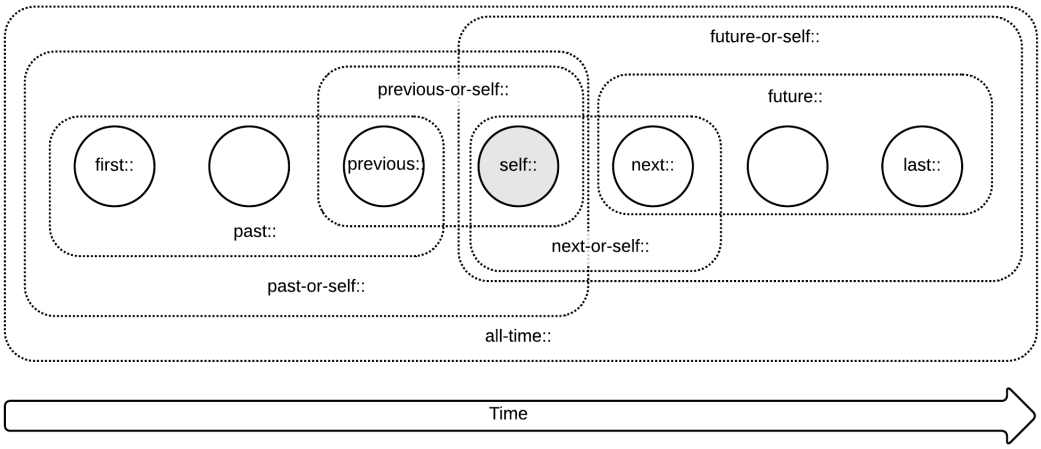


Figure 3.1: XPath temporal extension implemented in Sirix

3.2 XML Differencing and Merging Tools and Formats

Regarding the differencing and merging tools and formats, there are numerous alternatives available. Lindholm et al [36], La Fontaine [37], Rönnau [27], Komvoteas [38], Y. Wang et al [39], F. Wang and Zaniolo [31], among others, all present different XML differencing algorithms. Further more, Rönnau [40] Lindholm et al [36], and Y. Wang et al [39], perform comparisons between alternative differencing tools.

In Chien et al [41] they compare the use of versioning schemas for XML with revision control system and Source Code Control System, concluding that both "RCS and SCCS are not up to the task and there is a need for new and improved techniques that achieve better performance at the physical level and the logical level." [41].

3.3 Versioned XML Storage

3.3.1 TreeTank

Graf [42] and Graf et al [43] present an architecture for a versioned XML database called Treetank[44]. Its architecture is based in three core concepts, these are: "the nodes must contain all information about their content and their position in the overall structure", "the position of a node in a tree must be flexible regarding its position in the storage", and that "changes to the stored nodes must rely on a convenient and confident transaction system".

3.3.2 Sirix

Lichtenberger, originally involved in the group that created TreeTank [43], creates Sirix[25] as a fork from TreeTank in a followup to his MSc Thesis[35], in order to maintain focus on Version control of XML as the previous project had shifted its towards secure cloud storage [45]. Lichtenberger also added in the project brackit(.org) to the project. Brackit is a query engine for XQuery developed at the TU Kaiserslautern.

4

Research Method

This chapter describes the research method followed to execute this thesis work. In the first section it is presented the overall research approach followed in this work. The second section presents the method followed to answer RQ1 by eliciting and specifying the required features. The third section presents the Feature Analysis performed to answer RQ2. The fourth section presents the experimental protocol followed to perform the Benchmark Analysis required to answer RQ3.

4.1 Approach

This work focuses on two different paradigms on XML version control (XML-aware and non-XML-aware) and performs a comparison between them. To achieve this goal, and in an initial stage, a literature review was performed to allow the understanding of the state of the art as well as a few workshops were made in conjunction with the supporting company in order to define and understand the magnitude of the problem.

The next stage, and after the problem has been more clearly defined, usage scenarios were elicited from the experts at the supporting company. Then they were specified and, after that, prioritised by the experts in a different workshop. At this stage, the functional and knowledge gaps were more clear and also understood where the current technologies in use weren't fulfilling the users' needs.

Given the two major possible paradigms that we've defined, the next stages focus on the development and evaluation of at least two solutions (one for each paradigm) on a prototype level.

For the non-XML-aware, the main idea was to extend and improve the existing technology through minor modifications (normalisation of input and output) that should increase the effectiveness of these tools with a low cost of implementation in some of the scenarios where these tools currently don't hold. Regarding the XML-aware tools, the

focus was on exploring existing solutions that could potentially address the problem and select one to implement and evaluate.

For the demonstration of the solution and the evaluation of the results in a systematic way, a lightweight testing framework was designed and developed specifically for this purpose with the goal of automating most of the tests. Using the aforementioned testing framework along with some manual testing, we compare the solutions against the user scenarios and defined goals.

The final stage of this research takes place with the writing of the project report directed towards the supporting company, this thesis report, and the thesis presentation.

The general design cycle approach and the evaluation strategy are described in the following sections.

4.1.1 General Design Cycle

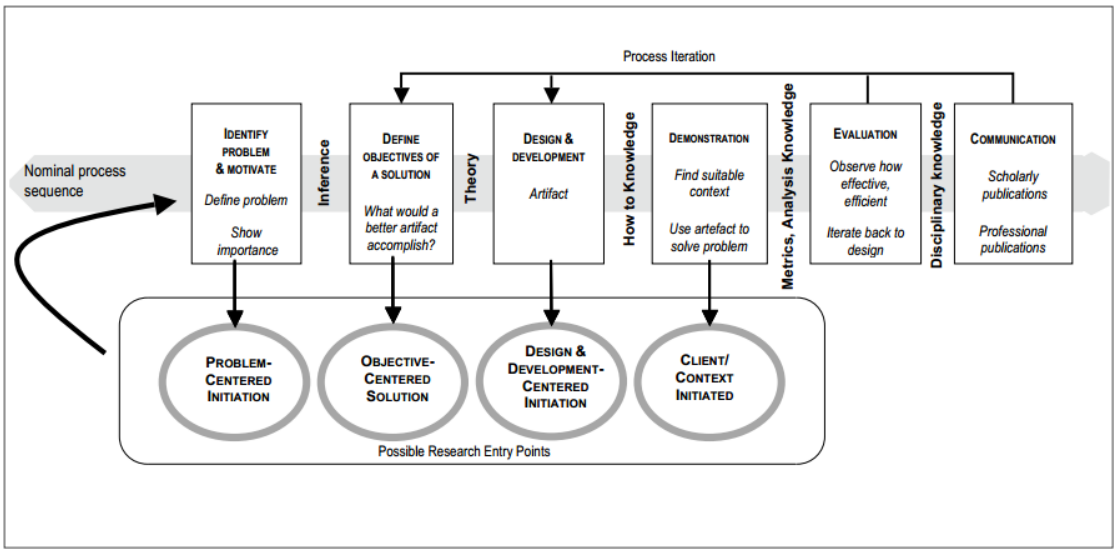


Figure 4.1: Design Science Research Methodology Process Model.[1]

Due to the nature of the proposed problem, this study follows a Design Science Research methodology [1, 46] in a non-strict way, i.e. the process was adapted to the problem while still keeping a close relation with the methodology. The selected general design cycle as figure 4.1 shows, is then divided into six separate stages: Problem identification and motivation; Definition of the objectives for a solution; Design and development; Demonstration; Evaluation; and Communication. Note the iterative process that allows for refining the outcomes of each stage with the exception of the problem identification and motivation. Each of these stages is described in the following paragraphs.

The actual workflow followed along with its relation to the aforementioned methodology is present in figure 4.2 on the following page.

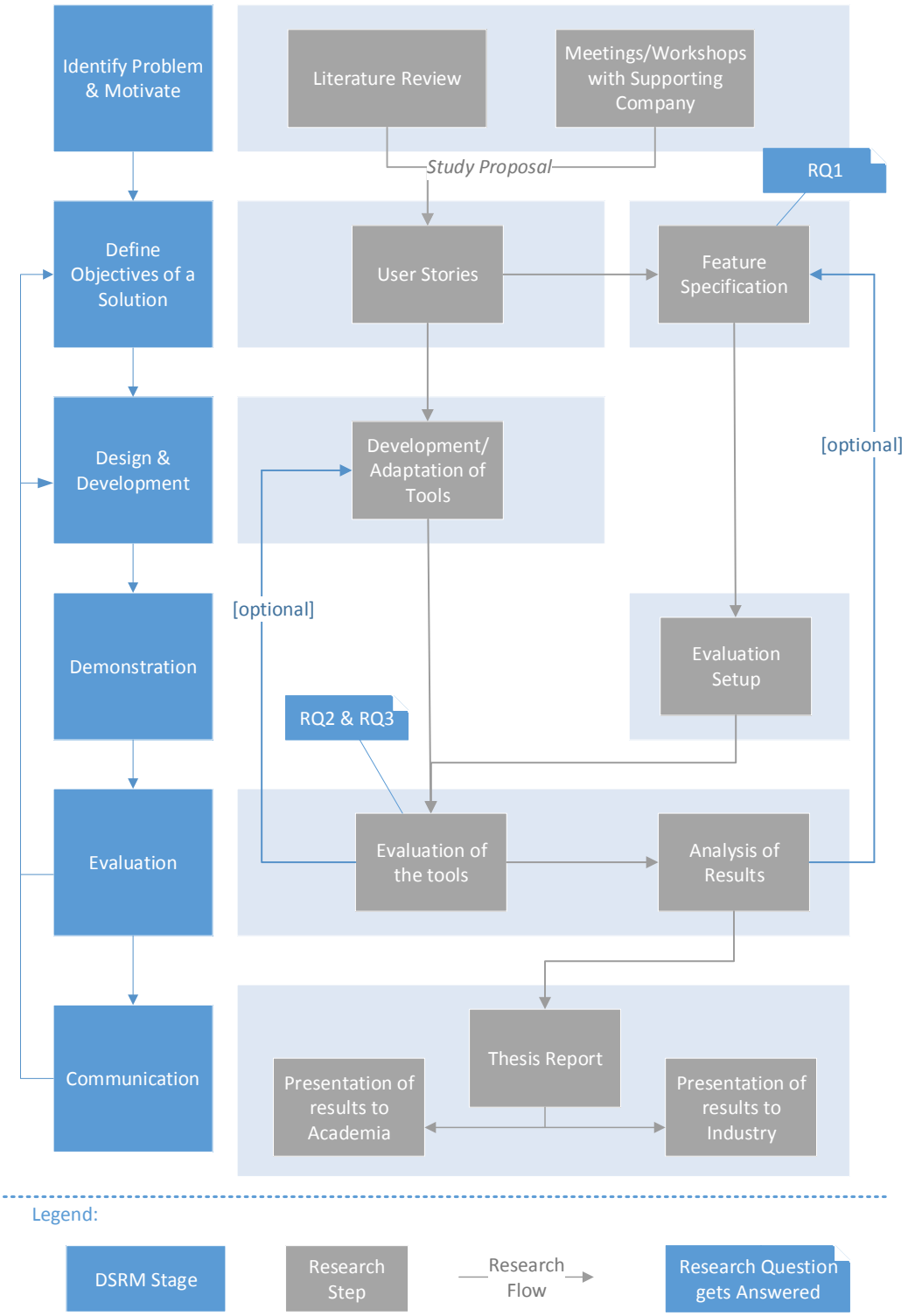


Figure 4.2: Thesis Workflow

Problem identification and motivation

During this phase, meetings with the supporting company took place concerning the problem definition. A literature review was also performed on the matter in order to better understand the state of the art regarding the subject. The main outcome of this activity was a study proposal, where the research questions were defined and motivated for.

Define the objectives for a solution

In this activity usage scenarios defined, user stories written and features for the tools were specified. These features are measurable either qualitatively or quantitatively. It is at the end of this stage that the Research Question 1 gets answered.

Design and development

This stage is self-explanatory, here the actual design and development of the prototypes of the tools were done to address the goals.

Demonstration

In the demonstration phase, or evaluation setup, activities such as experiments, simulations, etc. took place, these enabled the subsequent evaluation phase in order to verify that the tool did indeed solve the problem.

Note that the framework (DESMET) that supports this and the following stage of the cycle (Evaluation) is better described in sub-section 4.1.2 on the next page.

Evaluation

The evaluation stage concerns the measuring of "how well the artifact supports a solution to the problem" [46]. This activity involves the evaluation of the testing results against the features brought up by user scenarios, other quality metrics that have been defined, the feedback from the supporting company, among others.

At the conclusion of this phase, the research as a whole was re-evaluated and allowed to iterate back to either the objectives definition phase — in case of the features specification — to the design & development phase — in order to improve the tool — or, instead, proceed to the communication phase.

Before the transition to the next stage, the remaining Research Questions(2 and 3) must be answered.

Communication

Based on the knowledge collected in the above mentioned stages, in this last step two separate communication sets were produced. The first, regarding the supporting company, a project report was written along with a presentation performed; The second, in regards to the academia, this thesis report was written and the thesis defence took place.

4.1.2 Evaluation Strategy

Albeit we are at some point comparing *apples and oranges* as these tools have different approaches towards the same target, the end goal is to know which paradigm has the best cost benefit, or even if they are both mature and ready to be *consumed*. So, in order to be able to perform the comparison between the two approaches, a fair evaluation criteria has to be set and defined.

Kitchenham [47, 48] introduces an evaluation framework (DESMET) and a procedure to select the most appropriate evaluation method according to circumstances. The Feature Analysis and Benchmarking are deemed as appropriate means to perform a comparison between two or more alternate tools in our case.

In order to delimit the amount of tools to be reviewed and compared, an initial Feature analysis/Qualitative screening [47] was performed. This allowed to narrow down to the more relevant tools to later elaborate on the comparison.

Qualitative Screening

Kitchenham[47] defines qualitative screening as "a feature-based evaluation done by a single individual (or cohesive group) who not only determines the features to be assessed and their rating scale but also does the assessment. For initial screening, the evaluations are usually based on literature describing the software methods/tools rather than actual use of the methods/tools."

In order to answer the Research Question 2 — to know what features do these tools have — the Qualitative Screening evaluation method was selected.

Qualitative screening - Superficial setup

In order to perform this qualitative screening, an initial requirements elicitation was performed to understand the needs for the tools. These are then specified and detailed into measurable metrics.

Following that, a broad search was performed to gather the relevant available tools, both academic and industrial literature was reviewed with a focus on specialised conference proceedings (XML Prague)[49–53] to understand the specific tools that were in the market. Along with that, references to some tools were also given by the experts at the partner company which enabled the collection of a substantial amount of tools to be screened.

It is then accessed the extent to which these tools support the features by reviewing promotional material or academic literature of each tool. After scoring them, a list is compiled and shortened through an elimination process. The tools that are part of this shortlist are then to be used on the next stages of the evaluation.

Qualitative screening - Detailed setup

In the second stage in the screening, each tool from the shortlist is then implemented and a deeper assessment is then performed. This assessment is done against a refined set of

rules for the level of acceptability for each metric. The framework for the establishment of the set of rules and their acceptability ranges was inspired in much by the work of [54], where it "proposes a framework of Critical Success Factors(CSFs) that can be used to manage IS [Information Systems] integration projects". The results of this stage are later reviewed, analysed and summarised in a score sheet.

Benchmarking

Kitchenham[47] defines benchmarking as "a process of running a number of standard tests using alternative tools/methods (usually tools) and assessing the relative performance of the tools against those tests".

This Benchmark analysis was deemed appropriate to provide an answer to the Research Question 3 — How much overhead does an XML-aware tool carry when compared to a non-XML-aware one?

In order to perform this benchmarking, a test suit was developed (described in detail in section 5.1 on page 31) to automate the tests and reduce the human factor from the analysis. The metrics deemed relevant for this analysis are Time, Memory, Disk Space and Processing Consumption.

A workshop destined to understand realistic testing scenarios was conducted with the experts at the supporting company. Based on that, and the input from our academic supervisor, a decision of gathering an reliable open source documentation project with a reasonable level of commits was taken and, after extracted and preprocessed, was used as input for the benchmark analysis. The results are then added to a score sheet, for later analysis.

4.2 RQ1: Data Collection – Features Specification

In order to answer RQ1 – "What features are required for a version control system to have in a documentation context?" – a subset of Requirements Engineering processes were followed that can be organised into four different stages: Elicitation, Analysis, Specification, and Validation.

4.2.1 Elicitation

To elicit the requirements for the system, the knowledge available from the experts at the supporting company was used, along with the literature review performed.

Workshops

Experts' profile The supporting company provided the authors with access to two experts in the structured documentation field. Both experts are Software Engineers, holding more than 15 years of cumulative technical experience within the field of structured documentation – specifically in version control and developing of authoring tools. These experts are co-authors and maintainers of a Document Management

System (DMS) that has been in the market for several years, having gathered along the years valuable information from its users adding up to their technical expertise.

The experts at the supporting company functioned as user representatives in a requirements elicitation workshop that spanned for multiple sessions, helping the authors identifying and refining the requirements for the system.

4.2.2 Analysis

After the elicitation stage, User Stories were written (see appendix A on page 72) and brought back to the experts for prioritisation, allowing for both validation and refinement of the elicited requirements.

4.2.3 Specification

In this stage the User Stories were converted into measurable features (see table 6.2 on page 38) and grouped into categories (see table 6.1 on page 41).

4.2.4 Validation

Along with the previous preliminary validation performed by the experts when prioritising the User Stories, these were further validated in the later Feature Analysis evaluation stage (see section 4.3) of the research through the feature testing of the tools.

4.3 RQ2: Data Collection – Feature Analysis

This section contains the description of the method used for the Feature Analysis performed in order to answer the second Research Question. Firstly the criteria used is described(in 4.3.1), followed by the scoring methods used for the features considered(in 4.3.2), and lastly, three different analysis examples are provided – each with different characteristics and complexity (in 4.3.3, 4.3.3, and 4.3.3).

4.3.1 Feature Analysis Evaluation Criteria

Feature Analysis: Screening mode

As described in sub-section 4.1.2 on page 22, the evaluation started by the screening of possible tools to take up to the next phase. For this screening, our criteria was quite high level, with some being objective (e.g. licenses, access to source code, novelty), and some being subjective (e.g. recommendation from the experts).

The final outcome of this pre-selection was the short list of four different tools to experiment with:

Git next to SVN, one of the *defacto* standard tools for versioning text files, selected mostly due to our own previous experience with it.

Normalised Git implementation of a normalising step on input before using a line based versioning tool.

XChronicler based on the previous work from [34], it was a recommended tool from the experts at the supporting company.

Sirix the candidate that seemed to have the best potential to be a full versioning system for xml.

Feature Analysis: Detailed mode

DESMET[55] suggests a set of thirteen different top level features to be evaluated:

1. ***Supplier assessment***
2. ***Maturity of method or tool***
3. *Economic issues in terms of purchase, technology transfer costs and cost of ownership*
4. ***Ease of introduction in terms of cultural, social and technical problems***
5. *Eligibility for required application areas*
6. ***Reliability of the tool software***
7. *Robustness against erroneous use*
8. ***Effectiveness in current working environment***
9. ***Efficiency in terms of resource usage***
10. *Elegance in the way certain problem areas are handled*
11. *Usability from the viewpoint of all the target users in terms of learning requirements and "user-friendliness"*
12. ***Maintainability of the tool***
13. ***Compatibility of the method and tool with existing or proposed methods and tools.***

Out of these suggested top-level features, eight have been selected (highlighted in bold) as relevant for this analysis based on the experts opinion.

These top-level features, combined with the user stories (full list in appendix A on page 72), resulted in the complete feature list presented in table 6.2 on page 38.

4.3.2 Feature Analysis Scoring

Judgement Scale

For each of the features mentioned above, a set of different criteria was developed to evaluate each metric. These criteria should have the following characteristics: be replicable; easily understandable; distinct, i.e. without possible fuzzy results; and quantifiable;

We then define the possible results on a scale from 1 to 5, where 5 is the best outcome, and 1 the worst.

Depending on the type of feature, the scale can have 3 different possible distributions:

1. Binary/Boolean (for binary qualitative results): {1, 5}
2. Ternary (for qualitative results): {1, 3, 5}
3. and Quinary (for quantifiable results): {1, 2, 3, 4, 5}

Fitting the score

After setting the criteria from the Judgement Scale step, and knowing the type of progression of the possible result of the feature evaluation, the score is then adapted to fit the scale defined previously. In most cases the fitting formulation followed a logarithmic progression. The following presents an example of this type of formulation:

1. < 100
2. < 1000
3. < 10000
4. < 100000
5. ≥ 100000

The remainder fitting formulations can be found in appendix B on page 75.

Score

After fully establishing the criteria, we then proceed to assess the score, which, depending on the type of criteria can be then calculated, verified by experimentation, or based on the tool's literature. The outcome of this step is then documented in a table like the one in table 4.1.

Table 4.1: Scoring example

Scoring example					
id	**Metric Name**	**Git**	**Normalised Git**	**XChronicler eXist**	**Sirix**
Id	Name of feature	[1-5,N/A]	[1-5,N/A]	[1-5,N/A]	[1-5,N/A]

4.3.3 Feature Analysis Examples

In this section, three different examples of the feature scoring are presented, the remaining detailed metrics and results are presented in appendix B on page 75.

Example 1 – MCR-04: Handling of non-signicant white-space

Description: *MCR-04: Handling of non-signicant white-space* is the capability that the tool has to ignore the existence of non-significant white-space within the XML document(see sub-section 2.4.4 on page 13).

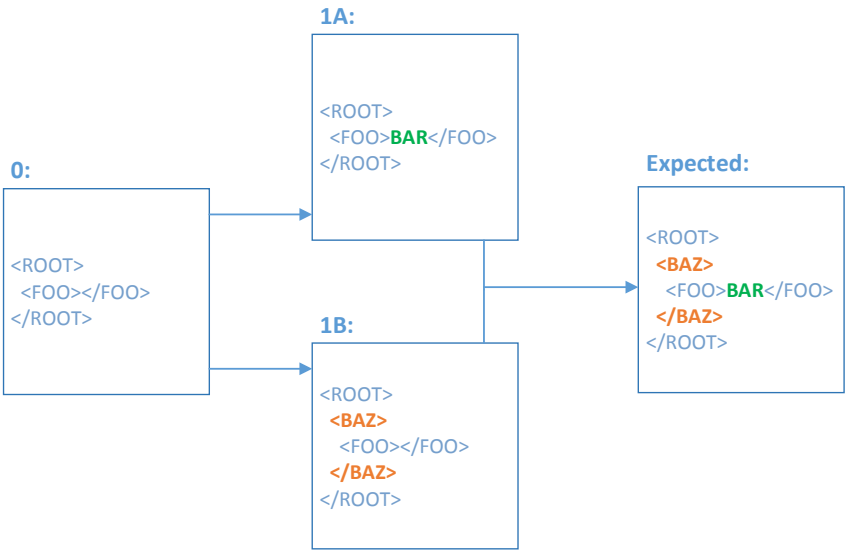


Figure 4.3: Example of test design for evaluation of XML-03

Formulation: This feature is evaluated in a binary/boolean scale, where:

- 5 - Exists
- 1 - Does not exist

Example 2 – XML-03: Handling insertion of parent

Description: *XML-03: Handling insertion of parent* evaluates the capability of the tool to handle the insertion of a parent above one (or many) elements – e.g. in the context of documentation, to create a section that contains already existing text as in figure 4.3.

Formulation: this feature is evaluated in a ternary scale, where:

- 5 - Possible and can not create unintended conflicts
- 3 - Possible but can create unintended conflicts
- 1 - Not Possible

Unintended conflicts are conflicts that are not expected to exist. E.g. given a base element, when merging a commit A – that adds a text content – with a commit B – that adds a parent to the base element – there should not be a conflict and the merge should run smoothly.

Example 3 – PM-03: Level of Documentation of the Project

Description: *PM-03: Level of Documentation of the Project* evaluates the availability and type of documentation of the development project of the tool.

Formulation: In order to assess the Level of Documentation, and given the natural fuzziness of this type of feature, we include a set of different parameters into the equation, each with different weights. The considered formula was the following:

$$\begin{aligned} x = & \textit{Book hits} \times 100 + \\ & + \textit{Search Engine hits} \times 0.01 + \\ & + \textit{Expert Community Q\&A hits} \times 1 + \\ & + \textit{Source Code lines of comment} \times 0.01 + \\ & + \textit{Official Documentation lines of text} \times 0.1 \end{aligned}$$

Where:

Book hits number of books hits found on a major online book reseller (i.e. amazon.com) for the name of the tool and the works "version control".

Search Engine hits number of hits on the major online search engine (i.e. google.com) for the name of the tool and the words "version control".

Expert Community Q&A hits number of questions tagged on the major online software development expert community Q&A (i.e. stackoverflow.com).

Source Code lines of comment number of lines of comment within the source code. For this analysis, a source code analysis tool (ohcount[56]) was used.

Official Documentation lines of text In order to measure the amount of official documentation available, we opted for selecting the number of lines of text in the official documentation.

Given the different type and amount of information usually available within each of the above described parameters, their contribution to the end score differs.

Taking the *Expert Community Q&A hits* as the baseline, the other features were weighted according to the estimated amount of information on a 10^n based scale. All the data has been collected in 2015-04-10.

Given that it is a quantifiable results we selected the quinary ($\{1, 2, 3, 4, 5\}$) scale for scoring the results.

Fitting the score: Given the expected range of results, a 5-stepped logarithmic scale (starting at 100) was selected.

4.4 RQ3: Data Collection – Benchmark Analysis

A subset of the OpenStack Manuals[57] was selected as the data used in the performance tests.

The selection of this dataset took into account the following criteria:

- Popular repository
- Different repository contributors
- Easy access to repository history
- Reasonable documentation size
- Formatted in well-formed XML

4.4.1 Pre-processing

In order to facilitate the insertion on the XML-aware approach, we tuned the data in order to exclude features that were not within the scope of the study. This included the removal of elements such as x-links, x-includes, namespace declarations, and processing instructions, along with special characters.

After this setup we ended up with 30 unique files and a series of 129 unique commits.

4.4.2 Execution steps

The following list presents the execution steps for the performance evaluation, see appendix E on page 84 for implementation details.

1. Create a folder for each file and put all the versions of that file in that folder.
2. For each file in the above, remove comments, processor instructions, xlinks, and special characters.
3. For each folder run the init command on the common-api
4. Send in all the files as commits to the common-api
5. Read the output that contains time, memory, CPU, and hard-drive space.

4.4.3 Post-processing

From the extracted data, the Memory consumption which is in percentage relative to the test environment is then converted into absolute values before proceeding to the analysis of these results.

4.5 Approaches under test

For the benchmark analysis we needed an automated test suite for reducing the influences of the human factor. We decided to build this test suite in a modular form for later reuse of some of the modules and also to be flexible in case we need to improve, we can iterate back and change or rewrite a part or change the selected tool.

4.5.1 Git

Currently Git is arguably the most advanced and used tool for versioning source code, it then comes naturally as a proper baseline to experiment with.

4.5.2 Normalisation of XML Input

non-XML-aware tools, are unable to accurately perceive context and to identify when a change has been performed. With the distributed environment and possibly different tools editing the same XML files, some changes might occur that are not actually structural changes in the XML, but are perceived as such by line differencing tool. An example of that would be the change of the order of the attributes of a node, this would still represent the same node with the same characteristics and would not represent a change in an XML-aware tool, but for a line-based differencing tool it would be perceived as a change.

A possible way to counter these limitations is to then force the input to be normalised and to always follow a strict set of rules with regards to its form.

The selected normalisation steps were to first perform an XML C14N[9] (see subsection 2.1.3 on page 7), followed by splitting the attributes of a node into their specific rows. This allows many of the common non-content changes to be ignored by the versioning tool while increasing the change detection within attributes of an element.

4.5.3 XML-aware Versioning (Sirix and XChronicler)

We implemented XChronicler with a different backend for meeting the needs of the API for testing purposes, we also wanted to be able to update current generated *v-files* so we added an update function.

Regarding Sirix' evaluation, given that it is missing the branching model, and in order to be able to test the built in merging ability, we implemented an extration method that extracts a patch in the form of an XQuery Pending Update List, the decision for using Pending Update Lists was made based on it being part of the XQuery Update Facility[15] recommendation from W3C.

5

Implementation

For the evaluation, a series of adapters, tests and minor features were developed, this chapter presents the most important implementation details. We start by providing the architectural overview of the test framework, and then go into details on the specific modules developed.

5.1 Overview

The test environment has three major components, the Test GUI, the Common API, and the Test API.

The Test GUI presents the interface (webpage) towards the tester, it allows to select which tests to run and retrieve its results.

The Common API layer is an interface that sets the features that each tool should comply with, allowing for a more fair evaluation of the different tools.

The Test API functions as a bridge between both the test database and the Test GUI.

In figure 5.1 on the next page, one can see this high level structure of the implementation of the test framework.

5.1.1 Usage scenario

A typical usage scenario of the test framework can be seen in figure 5.2 on page 33, where a user selects a test and executes it, and the results are then stored in the *Test DB*.

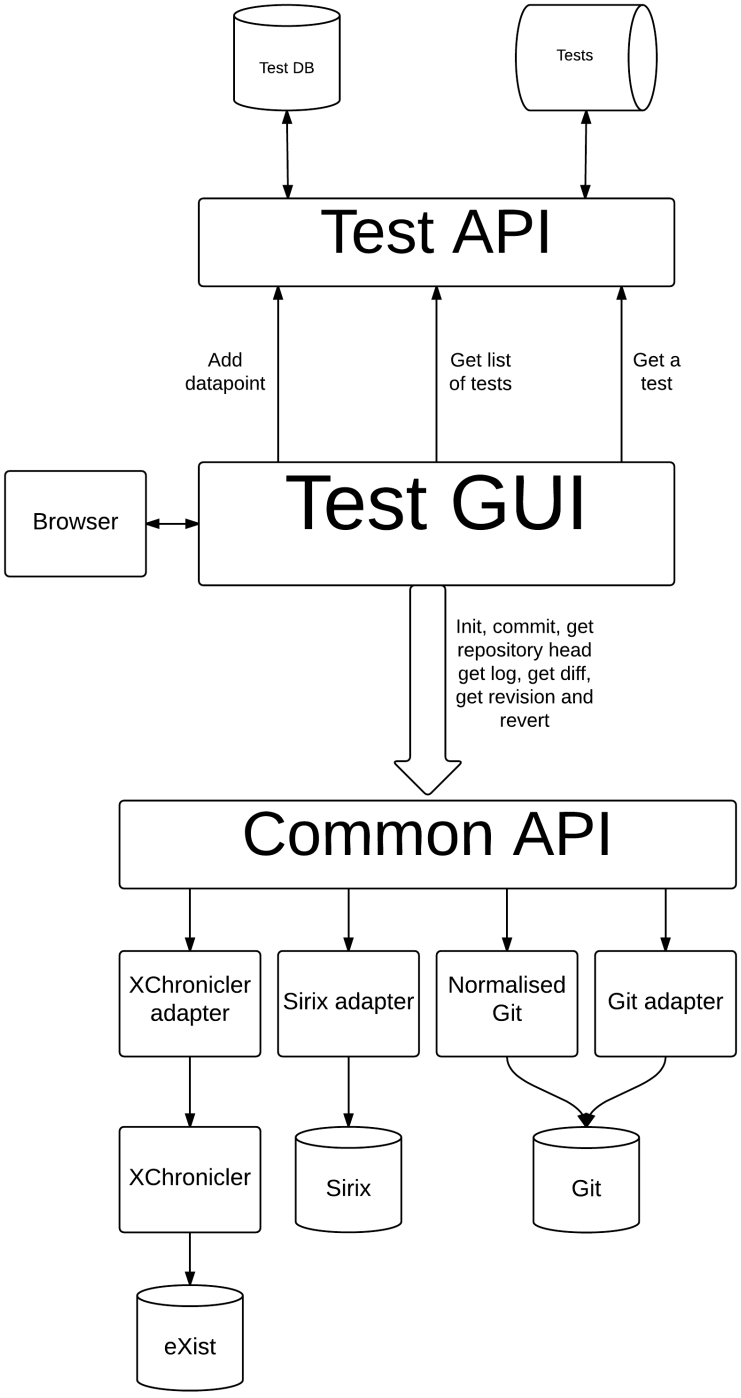


Figure 5.1: Structure diagram of the test environment architecture.

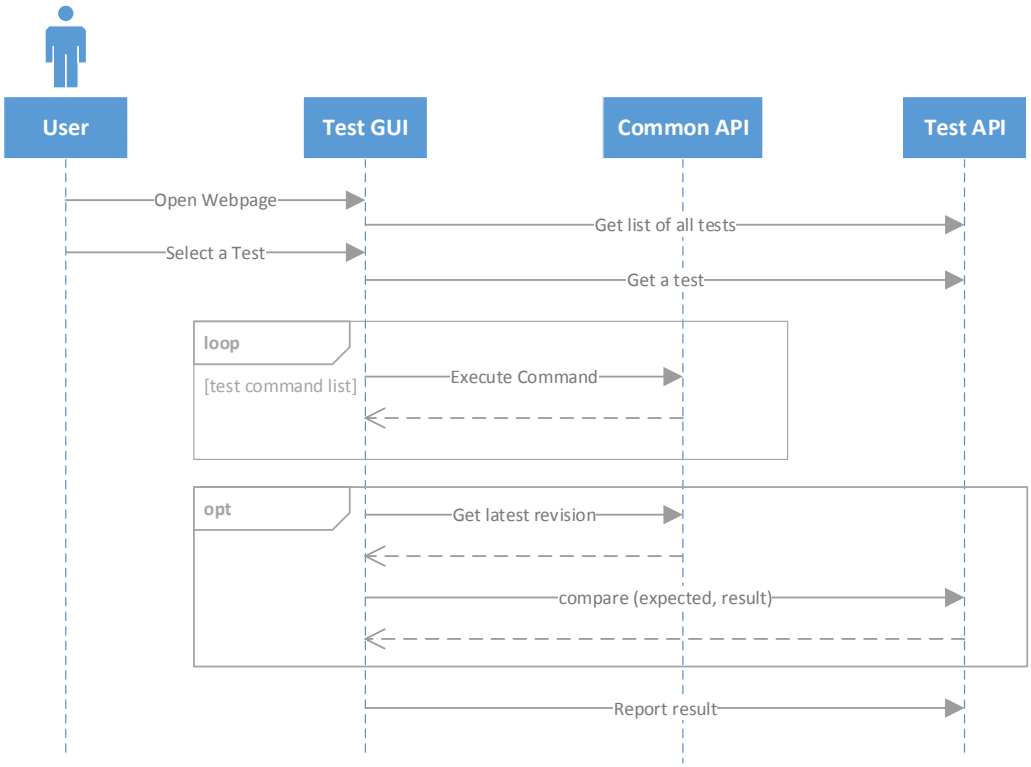


Figure 5.2: Testing Framework – typical usage sequence diagram

Usage Example

Use Case: "More than one person editing the same file in different parts at the same time".

How to perform the test:

1. The user selects the platform to which it runs the test against (e.g. Normalised Git).
2. The user selects the test to run, in this case the test *ParallelEditWithoutConflicts*
3. The *Test GUI* then gets the test script from the *Test API*, and starts executing it against the *Common API*.
4. The results then are reported to the *Test DB* for later processing during the evaluation phase.

This concrete test scenario script can be seen in the following list and its implementation can be found in appendix G on page 87.

1. *Pre-conditions:* Repository, File, and File content exists

2. Alice checks out repository
3. Bob checkouts repository
4. Alice edits file
5. Bob edits same file in different place
6. Alice commits changes
7. Bob commits changes
8. *Post-conditions:* Resulting file is the correct merge of the 2 commits with the original file.

5.1.2 Language and frameworks

For the server side language Test API and Common API the selected language is java, due to its usage within the supporting company's servers and the authors are well versed in it.

The server side implementation uses Java Play[58] framework, due to its REST capabilities.

For the client-side language, JavaScript is used. No client-side framework was used for flexibility reasons.

5.2 Common API

The Common API is an interface for the different versioning tools. It lays on top of the different adapters that implement this interface, this design allows for an easy usage of different tools without touching the main modules.

The different layers communicate between themselves through AJAX requests, that are then interpreted in JSON within each module.

The Common API has following interface:

init Creates a new repository, removing the previous one.
commit Attempts to commit a file or a Pending Update List list to the latest revision of the repository or to a determined version, if specified.
getRepositoryHEAD Retrieves the latest version of the repository.
getLog Gets all the commit IDs with their messages.
getDiff Sends in a relative version and receives diff script as response: a Git diff in case of Git, and a Pending Update List in Sirix's case.
getRevision Sends an ID of a specific revision and receives its content.
revert Reverts the repository to a specified version.

5.3 Git

For the implementation of the Git module, Eclipse Foundation's JGit[59] was used out-of-the-box due to its native java form.

5.4 Normalised Git

For the implementation of the Normalised Git, the process was divided into two separated phases. The first, where the C14N happens, and the second where some further normalising steps are taken.

5.4.1 Canonicalisation process

The C14N of XML (described in sub-section 2.1.3 on page 7) is performed with the help of an external tool (xmllint[60]) that transforms the input into a XML Canonical v.1.0[9] compliant version.

5.4.2 Further Normalisation

After the C14N, a simple set of rules are applied to the canonical XML. A new line is initiated after each attribute in an element, with the intention of making it easier to detect where changes occur, and also to avoid conflicts in concurrent changes in different attributes belonging to the same node.

5.5 XChronicler + eXist

In this implementation we extend Svallfors' work[34] and modify the storage medium from a regular file system to an XML Database with an XQuery processor (eXist)[22], allowing us to perform multiple queries on the data.

Despite our implementation efforts, the XChronicler + eXist implementation never reached par in terms of features with the other versioning tools, for more details, see discussion in chapter 7 on page 53.

5.5.1 XChronicler

This original implementation, as a a versioning system that creates a V-Document from two (or more) different versions of the same document.

Initially this implementation was not ready to expand the originally created V-Document with a new added version, since this was required to properly test the capabilities of this tool, this feature was implemented.

5.5.2 eXist

The eXist storage and XQuery processor, allowed us to perform a series of different queries that enabled the system to perform closer to an actual versioning system.

These queries allowed to retrieve different revisions of a document from the generated V-Document, along with retrieving the list of revisions.

5.6 Sirix

For the implementation of the Sirix module, a set of XQuery instructions were used to perform the required actions.

Regarding the creation of patches, a Pending Update List parser was developed, this translated the Java objects that were returned from Sirix's engine when creating a diff, into Pending Update Lists.

These Pending Update Lists were then used to recreate the different versions of a file.

Although Sirix has a multiple interface – i.e. Java Objects, REST, and XQuery – the decision of using Pending Update Lists and the XQuery expressions as the way to communicate changes to the engine was due to the standardised approach that these convey.

Along with the Pending Update List parser, some headers were added to the documents, allowing the required queries to be performed, the headers added were the *commit message*, *author*, and *timestamp*.

5.7 Data collection for benchmark analysis

All the tested tools used the same process for data collection, as described next.

5.7.1 Memory and CPU

For the collection of *Memory* and *CPU* usage, we use the application "top"[61] with the following arguments:

```
top -b -d 0.5 -n 100
```

Translating to run top in batch mode, updating every 500ms, and limited to 100 iterations — this translates into more than the necessary iterations for the test time, the excess data is then removed in a post-processing step.

5.7.2 Time and Repository Size

For the collection of *Time*, Java's built in API was used with the following function call: `System.nanoTime()`

Lastly, for the *Repository Size*, the Java's API built in function for getting file size was used: `File.length()`

5.8 Source Code

The source code for the frameworks that have been produced for this thesis are released in github under the Apache 2.0 license[62]:

- The Versioning Framework is currently on 396e4430 and can be found in: `https://github.com/XMLVersioningFramework/XMLVersioningFramework`

- The Test Framework is currently on deae63ed and can be found in: `https://github.com/XMLVersioningFramework/XMLTestFramework`

6

Results

This chapter presents the findings of this research. It is divided in three major sections, one for each of the first three research questions. Firstly, the outcome of the requirement elicitation and the summary of the required features provide the answer to RQ1. Secondly, the results of the Feature Analysis provide the answer to RQ2. Thirdly and last, the outcome of the Benchmark Analysis provides answer to RQ3.

6.1 RQ1: What features are required for a version control system to have in a documentation context?

6.1.1 User Stories

The workshops and the literature review performed allowed to elicit some important user stories that were then validated by the experts, these are specified in appendix A on page 72 and are a byproduct of this research that can be used in future implementations.

6.1.2 Features

The user stories were then restructured into features and grouped by their nature as can be seen in table 6.1 on page 41. The fully extended feature list can be seen in table 6.2.

Table 6.2: Feature list

ID	Metric Name
Versioning Features	
V-01	Fulfilment of tracking of file changes

ID	Metric Name
V-02	Fulfilment of reversion of a file to a previous version of choice
Repository Management Features	
RM-01	Existence of different branches of the same repository
RM-02	Ability to tag a specific revision of a repository
RM-03	Ability to merge different branches of the same repository
RM-04	Ability to remove/untrack a file on the repository
RM-05	Ability to amend a commit
RM-06	Ability to rename a file
RM-07	Ability to fork a document
RM-08	Ability to handle different users
Merging / Conflict resolution Features	
MCR-01	Ability to handle conflicts
MCR-02	Ability to merge different patches
MCR-03	Ability to interactively merge hunks of a file
MCR-04	Handling of non significant white-space
Project Maintainability Features	
PM-01	Level of Popularity of the project
PM-02	Level of Activity of the project
PM-03	Level of documentation of the project
XML Specific Features	
XML-01	Handling insertion of child
XML-02	Handling insertion of sibling
XML-03	Handling insertion of parent
XML-04	Handling of element rename
XML-05	Handling of removal of elements
XML-06	Handling of move of an element
XML-07	Handling of attributes insertion
XML-08	Handling of attributes removal
XML-09	Ignores the change of order of an attribute
XML-10	Ability to enforce well-formed xml on the repository

ID	Metric Name
XML-11	Ability to track changes on a node
XML-12	Ability to retrieve the history of a specific node
CMS Specific features	
CMS-01	Ability to retrieve list of changes (log)
CMS-02	Ability to retrieve list of modified elements in the current working copy
CMS-03	Ability to flatten a modular document
CMS-04	Ability to reuse parts of a file in others

6.2 RQ2: Of the previously identified features, which ones do XML-aware and non-XML-aware tools have?

6.2.1 Feature Analysis result – examples

Here the examples previously introduced in section 4.3 on page 24 have their results presented.

Example 1 – MCR-04: Handling of non-significant white-space

Regarding the *MCR-04: Handling of non-significant white-space*, Git is the only tool that is not able to handle it as there is no distinction on significant and non-significant white-spaces. The MCR-04 scoring sheet is in 6.3.

Example 2 – XML-03: Handling insertion of parent

In this example we look closer at the scoring and results of the feature *XML-03: Handling insertion of parent*.

Both Git and Normalised Git do not fully succeed in handling the insertion of a parent element. This can be verified by the implementation of the previous example test (see figure 4.3 on page 27) in figure 6.1 on page 42 where both trigger unintended conflicts as opposed to XChronicler and Sirix. The scoring sheet for XML-03 can be found in table 6.4 on page 42.

Example 3 – PM-03: Level of Documentation of the Project

Regarding the *PM-03: Level of Documentation of the Project*, the results vary greatly, with Git achieving the maximum result, XChronicler and Sirix the minimum, and Normalised Git being excluded from this evaluation as it was developed by the authors when performing this research.

Table 6.1: Feature groups description

Prefix	Category	Conceptual Definition	Metrics
V-00	Versioning	Comprises the basic features that relate to keep track of versions – these are mandatory and are meant to dismiss any non-fulfilling tool.	2
RM-00	Repository Management	Contain the file structure and its related features.	8
MCR-00	Merging/Conflict resolution	The features that relate to the tools ability to merge and solve differences in a file.	4
PM-00	Project Maintainability	Contains the characteristics of the project that supports the tool. Its goal is to provide a degree of reliability and trust to the tool.	3
XML-00	XML Specific	The features related to versioning that are unique to XML.	12
CMS-00	CMS Specific	Features that are not entirely scoped within the above categories and are not expected to be addressed by the existing solutions but are a 'nice to have' set of features.	4

The detailed scoring of the various parameters of the different tested tools can be seen in table format in table 6.5 on page 43, table 6.6 on page 44, and table 6.7 on page 44; and in chart format in figure 6.2 on page 43.

The resulting scoring sheet for this feature can be found in table 6.8 on page 44.

6.2.2 Feature Analysis Score sheets

The table 6.9 on page 45 presents the summarised version of the scores from the *Feature Analysis* evaluation grouped by *Feature Group*, the stacked area chart in figure 6.3 on page 45 presents a visualisation of this same data.

The detailed results follow this summarised view and are from table 6.10 on page 46 to table 6.15 on page 49. Each table contains a reference to the metric's *ID*, its *Name*, and the resulting *Scores* given to each of the appreciated tools.

Table 6.3: MCR-04: Handling of non-significant white-space results.

id	Metric Name	Git	Normalised Git	XChronicler eXist	Sirix
MCR-04	Handling of non-significant white-space	1	5	5	5

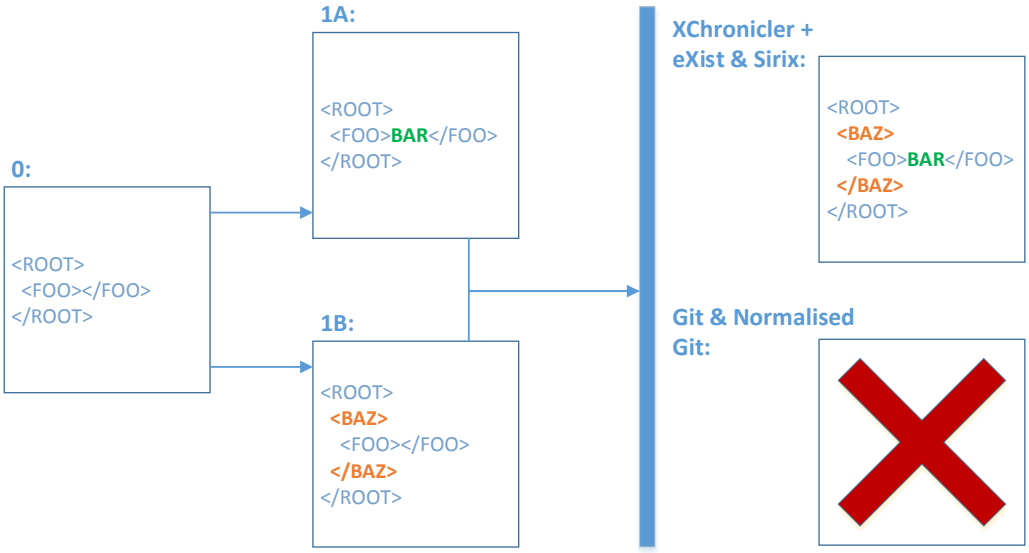
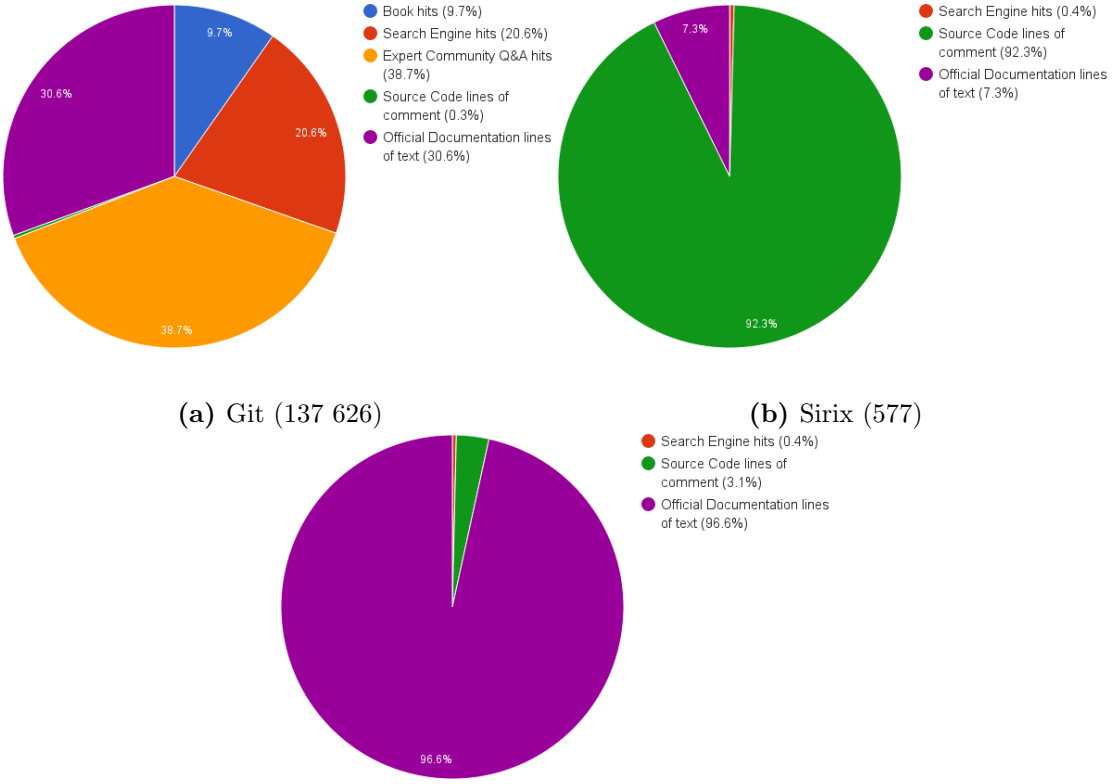


Figure 6.1: Example of test for evaluation of XML-03

Table 6.4: XML-03: Handling of insertion of parent results.

id	Metric Name	Git	Normalised Git	XChronicler eXist	Sirix
XML-03	Handling of insertion of parent	3	3	5	5



(a) Git (137 626)

(b) Sirix (577)

(c) XChronicler (121)

Figure 6.2: Level of documentation

Table 6.5: Git documentation comparison

Name	Value	Weight	Score
Book hits (9.7%)	134	100.00	13400.00
Search Engine hits (20.6%)	2840000	0.01	28400.00
Expert Community Q&A hits (38.7%)	53297	1.00	53297.00
Source Code lines of comment (0.3%)	42647	0.01	426.47
Official Documentation lines of text (30.6%)	421030	0.10	42103.00
Total			**137626.47**

Table 6.6: Sirix documentation comparison

Name	Value	Weight	Score
Book hits (0%)	0	100.00	0.00
Search Engine hits (0.4%)	239	0.01	2.39
Expert Community Q&A hits (0%)	0	1.00	0.00
Source Code lines of comment (92.3%)	53267	0.01	532.67
Official Documentation lines of text (7.3%)	419	0.10	41.90
Total			**576.96**

Table 6.7: XChronicler documentation comparison

Name	Value	Weight	Score
Book hits (0%)	0	100.00	0.00
Search Engine hits (0.4%)	45	0.01	0.45
Expert Community Q&A hits (0%)	0	1.00	0.00
Source Code lines of comment (3.1%)	372	0.01	3.72
Official Documentation lines of text (96.6%)	1172	0.10	117.20
Total			**121.37**

Table 6.8: PM-03: Level of documentation of the project results

id	Metric Name	Git	Normalised Git	XChronicler eXist	Sirix
PM-03	Level of documentation of the project	5	N/A	1	1

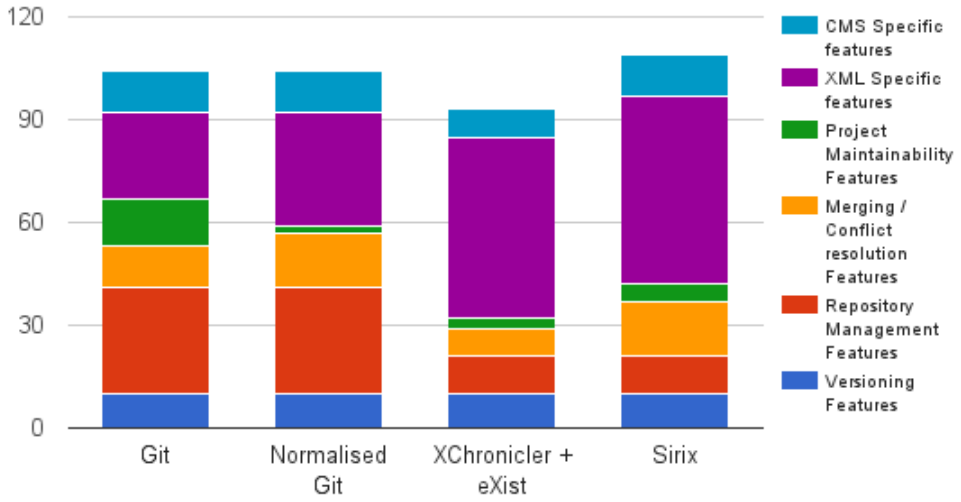


Figure 6.3: Feature Analysis results by Tool and stacked by Feature Group

Table 6.9: Summary of Feature Analysis results

Features	Maximum score	Git	Normalised Git	XChronicler + eXist	Sirix
Versioning	10	10	10	10	10
Repository Management	35	31	31	11	11
Merging / Conflict resolution	20	12	16	8	16
Project Maintainability	15	14	2	3	5
XML Specific	55	25	33	53	55
CMS Specific	20	12	12	8	12
Total	155	104	104	93	109

Table 6.10: Versioning Results

Versioning Results					
id	**Metric Name**	**Git**	**Normalised Git**	**XChronicler eXist**	**Sirix**
V-01	Fulfilment of tracking of file changes	5	5	5	5
V-02	Fulfilment of reversion of a file to a previous version of choice	5	5	5	5

Table 6.11: Repository Management Results

Repository Management Results					
id	**Metric Name**	**Git**	**Normalised Git**	**XChronicler eXist**	**Sirix**
RM-01	Existence of different branches of the same repository	5	5	1	1
RM-02	Ability to tag a specific revision of a repository	5	5	1	1
RM-03	Ability to merge different branches of the same repository	5	5	1	1
RM-04	Ability to remove/untrack a file on the repository	5	5	5	5
RM-05	Ability to amend a commit	5	5	1	1
RM-06	Ability to rename a file	5	4	1	1
RM-07	Ability to fork a document	1	1	1	1
RM-08	Ability to handle different users	5	5	5	5

Table 6.12: Merging / Conflict resolution Results

Merging / Conflict resolution Results					
id	**Metric Name**	**Git**	**Normalised Git**	**XChronicler eXist**	**Sirix**
MCR-01	Ability to handle conflicts	5	5	1	5
MCR-02	Ability to merge different patches	5	5	1	5
MCR-03	Ability to interactively merge hunks of a file	1	1	1	1
MCR-04	Handling of non significant whitespace	1	5	5	5

Table 6.13: Project Maintainability Results

Project Maintainability Results					
id	**Metric Name**	**Git**	**Normalised Git**	**XChronicler eXist**	**Sirix**
PM-01	Level of Popularity of the project	5	1	1	1
PM-02	Level of Activity of the project	5	5	1	2
PM-03	Level of documentation of the project	5	N/A	1	1

Table 6.14: XML Specific Results

XML Specific Results					
id	**Metric Name**	**Git**	**Normalised Git**	**XChronicler eXist**	**Sirix**
XML-01	Handling insertion of child	3	3	5	5
XML-02	Handling insertion of sibling	3	3	5	5
XML-03	Handling insertion of parent	3	3	5	5
XML-04	Handling of element rename	3	3	5	5
XML-05	Handling of removal of elements	3	3	3	5
XML-06	Handling of move of an element	3	3	3	5
XML-07	Handling of attributes insertion	3	3	5	5
XML-08	Handling of attributes removal	3	3	3	5
XML-09	Ignores the change of order of an attribute	1	5	5	5
XML-10	Ability to enforce well-formed xml on the repository	1	5	5	5
XML-11	Ability to track changes on a node	1	1	5	5
XML-12	Ability to retrieve the history of a specific node	1	1	5	5

Table 6.15: CMS Specific Results

CMS Specific Results					
id	**Metric Name**	**Git**	**Normalised Git**	**XChronicler eXist**	**Sirix**
CMS-01	Ability to retrieve list of changes (log)	5	5	1	5
CMS-02	Ability to retrieve list of modified elements in the current working copy	5	5	5	5
CMS-03	Ability to flatten a modular document	1	1	1	1
CMS-04	Ability to reuse parts of a file in others	1	1	1	1

6.3 RQ3: How much overhead does an XML-aware tool carry when compared to a non-XML-aware one?

The table 6.16 summarises the results of the Benchmark Analysis performed on the selected tools (Normalised Git and Sirix). The detailed results can be found in appendix C on page 81. The description of the environment used for this analysis is present in appendix F on page 86

Table 6.16: Summary of Benchmark results

	Sirix(avg)	**Git(avg)**	**Sirix/Git**	**Difference (Sirix−Git)**
CPU(%)	50.26	1.28	39.4/1	48.99
Overall CPU (ms × %)	36 186.70	154.71	233.9/1	36031.99
Hard-drive Size (kB)	973.69	7.40	131.6/1	966.29
Memory (GB)	0.74	0.53	1.4/1	0.21
Overall memory (ms × kB)	532.79	64.06	8.3/1	468.73
Time (ms)	719.99	120.87	6.0/1	599.12

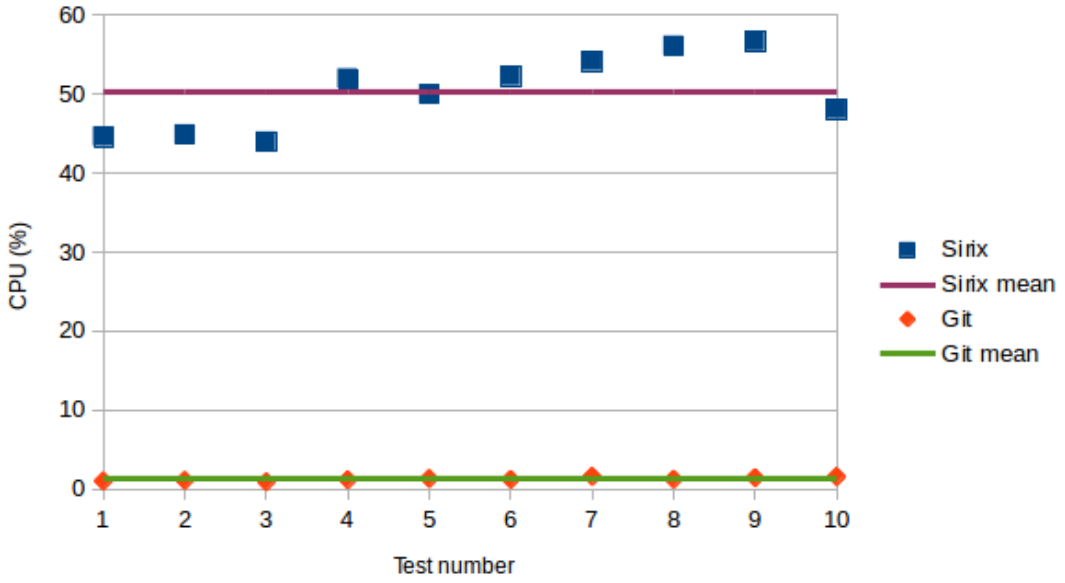


Figure 6.4: CPU Usage (less is better)

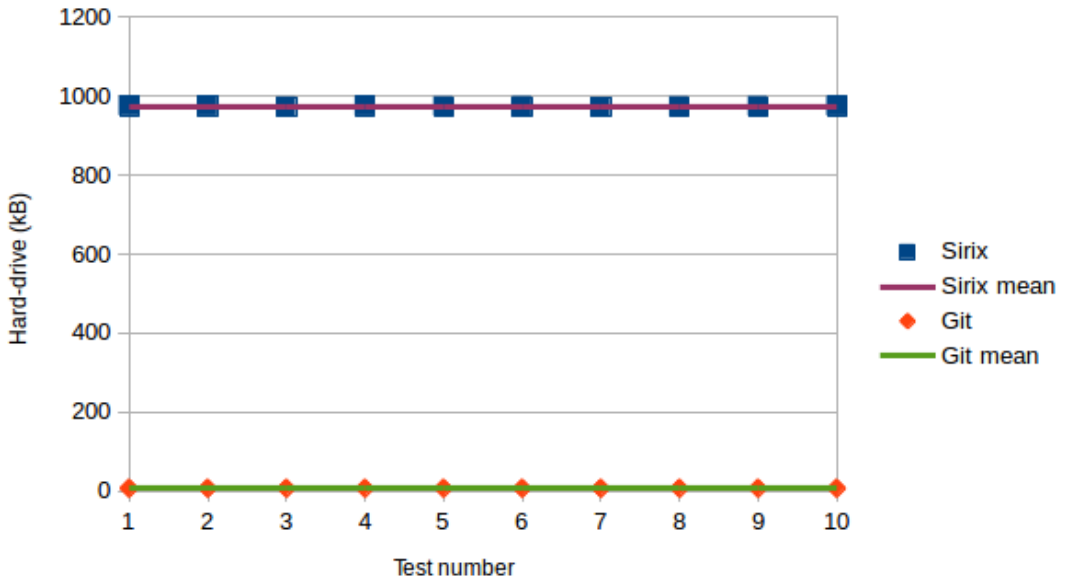


Figure 6.5: Hard-drive Usage (less is better)

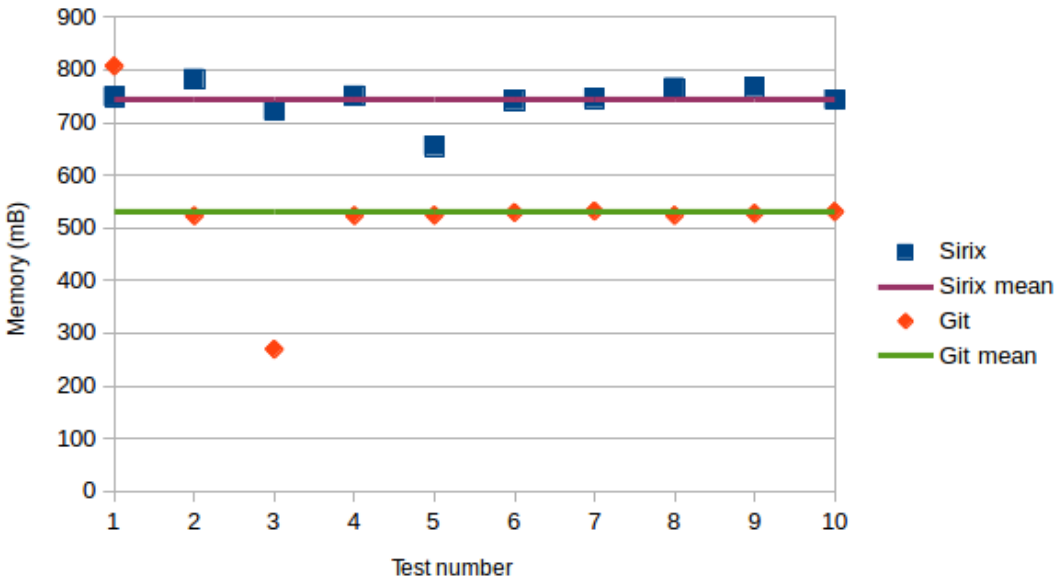


Figure 6.6: Memory Usage (less is better)

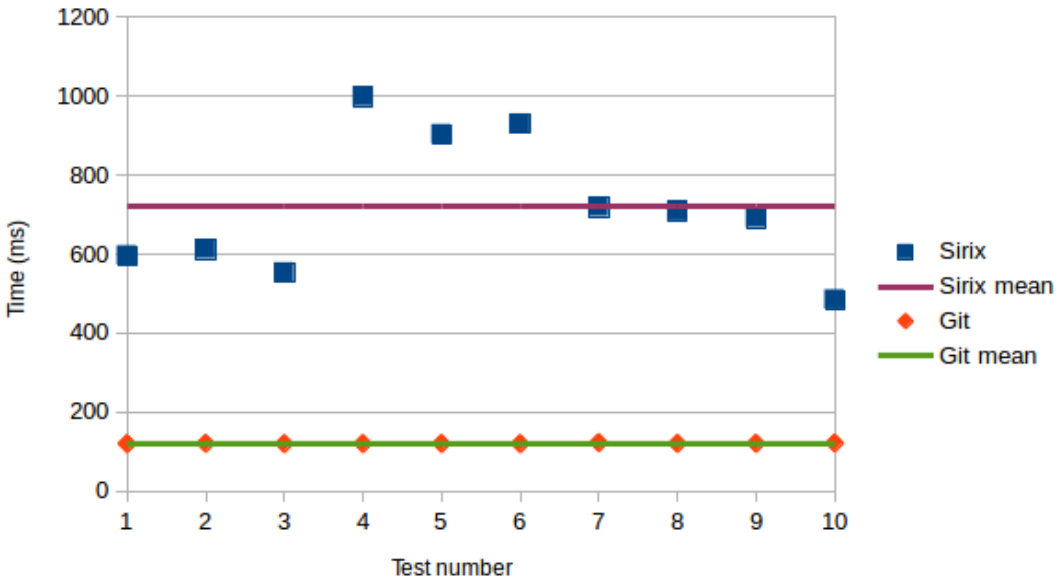


Figure 6.7: Time for the operation (less is better)

7

Discussion

In this chapter we discuss the results of the study. In the first section we address the individual research questions, followed by a discussion on the identified threats to validity and ethical conserns.

7.1 Research Questions

7.1.1 RQ1: What features are required for a version control system to have in a documentation context?

The elicitation stage started with several brainstorming sessions with the experts from the supporting company. Most of these sessions were unstructured and aimed towards the gathering of the experts' thoughts on the needs for the system they were envisioning.

This decision on the unstructured workshops approach was taken due to the scope not being clearly defined from the start, and the need to better understand the system that the experts envisioned given their several years of experience. Given both the technical and business profile of the experts, the need for (semi-)structured interviews was not deemed necessary by the authors.

Based on these sessions, a set of User Stories were written and brought to the experts for prioritisation and validation. During this period, there were still some uncertainties about the scope of the work, this required some iterations for the refinement of the user stories – the last version of these can be found in appendix A on page 72.

Before performing the transition to the formalisation of features, the authors, along with the academic supervisor scoped the work towards the evaluation of different approaches on versioning XML.

The formalisation of the Features (see table 6.2 on page 38) addressed the newly defined scope, selecting aspects from the user stories that were relevant to the research.

The validation of the Features was addressed by initially having derived them from the validated User Stories, and by having them validated during the Evaluation phase

of the research – during which the authors excluded non-quantifiable Features.

7.1.2 RQ2: Of the previously identified features, which ones do XML-aware and non-XML-aware tools have?

After gathering the necessary features, the initial screening process allowed us to narrow down the various available tools and further evaluate in detail a shorter selection of these tools/approaches.

After experimenting and attempting to fulfil the features with this narrower selection, we scored our experience in the score sheets present in section 6.2 on page 40, namely in tables 6.10, 6.11, 6.12, 6.13, 6.14, and 6.15.

Versioning Features

Regarding the *Versioning* features, there is no relevant differences as expected, as this was a basic requirement that the tools had to fulfil before being selected for the feature analysis.

Repository Management Features

For the *Repository Management* features, the non-XML-aware tools are much more evolved in this category. The Sirix project does contain many of these features on its roadmap to be implemented in the future, but these could not be taken into account as our consideration was on the currently implemented features only.

Merging and Conflict Resolution Features

In the *Merging and Conflict Resolution* category, the XChronicler+eXist tool fails considerably as conflict resolution is not within its scope. Otherwise, results between the different remaining tools are somewhat approximate with none really excelling. In the case of the document inserted being minified, the readibility of any conflicts occurring in Git is close to impossible to understand, with the pre-processing step added in Normalised Git cancelling this problem. However, with a 'pretty-printed' document, both Git and Normalised Git are much clear when throwing conflicts than Sirix.

Project Maintainability Features

For the *Project Maintainability* features, we analysed the project statistics, extracted from [63] and their own repositories [25, 64]. We considered the usage and compliance of established standards, its popularity and activity (contributors, commits, and supporting community), its maturity, and finally the quality and level of the project documentation available.

Unsurprisingly Git scores the highest in this category, achieving the maximum score possible. With the remaining projects, as expected, not having the same level overall.

To note that we excluded the Normalised Git project from the evaluation on the PM-03 (level of documentation) as it was developed by us. Still on this metric, it should be noted that, given the different distribution mean of the documentation of each tool (wikis, manpages, etc.), the extension of these might vary – not only due to the lack of content – but also probably due to differences in style inherent to the distribution mean.

Regarding the PM-02 (level of activity), we were initially inclined to select the number of line changes that had occurred in the repository during the last year, but given that a restructure in the folder structure of the project could create a massive number of line changes. We then opted for counting the number of commits as traditionally each commit portrays an activity step (bug fix, feature implementation, etc.).

Also of note is that on PM-01 (level of popularity), we use the number of stars a project has on Github to determine the level of popularity of a project. This is an arguable decision, but considering the huge popularity of Github, we believe it is a reasonable approximation to the popularity level of a project. Arguably, there are very few popular open source projects that are not present in Github.

XML Specific Features

Regarding the *XML Specific* features, the XML-aware tools score, as expected, much higher than the non-XML-aware ones. The non-XML-aware tools didn't achieve the best results, mostly due to the creation of unintended conflicts on some test scenarios. This occurs as expected (see sub-section 2.4.4 on page 13) due to their linear approach to context awareness.

It is interesting to highlight that there is a significant difference between Git and Normalised Git with the pre-processing step applied. By splitting the XML elements into several rows, we trick the context awareness of Git to accept merge commits with changes performed in different areas of the same element – e.g. when merging commits A and B, where A changes the value of an attribute, and B changes another, both originally in the same document line.

It should be noted that it is not expected that the *XML Specific* features not currently present in the non-XML-aware tools to be part of their feature roadmap anytime soon.

CMS Specific Features

In the last category of features, *CMS Specific*, both the non-XML-aware and the XML-aware tools achieve a bad score. This is mostly due to the fact that, as previously mentioned in sub-section 7.1.1 on page 53, these features are mostly out of scope from a standard versioning tool.

7.1.3 RQ3: How much overhead does an XML-aware tool carry when compared to a non-XML-aware one?

After having analysed the results from the previous question (summarised in table 6.9 on page 45), we selected the top scoring tools from each of the different approaches, namely

Normalised Git and Sirix.

These tools were then subject of a battery of tests designed to provide the benchmark results (summarised in table 6.16 on page 50).

For these tests, the input data used was a sample of the OpenStack Manuals[57] as mentioned in sub-section 4.4 on page 29, this documentation repository has a good set of characteristics that makes it a reasonable data input for the tests, such as the documentation structure, number of files, commits, and authors. Despite its quality characteristics, the sample documentation could not be processed in its whole due to some well-formedness(see sub-section 2.1.2 on page 7) errors with specific files/revisions. When facing these errors, the decided approach was to discard the whole file.

Git as a non-XML-aware tool was highly superior to Sirix in this benchmark analysis – where, besides consuming much fewer resources, it completed its tasks in 1/6 of the time of its competitor. These results confirm our expectations, as Git is a more mature and optimised tool, it doesn't have the expected overhead that an XML-aware tool would carry.

7.1.4 RQ0: What is a good approach to versioning an XML document?

Based on the results of the previous questions, our reasoning is that the final decision on a versioning tool is depending on many factors. As seen in the RQ2 (Feature Analysis) results, the analysed tools perform quite differently in the various categories. When choosing a tool or a type of approach, one must consider the overall needs and abilities of the system.

Assuming the more collaborators participate in editing a document, the probability of these editing the same areas of a document increases, and, that these edits will increase the probability of merge conflicts. Then, it is reasonable to assume that a tool that raises less collision problems will be able to handle more collaborators. Therefore, if the number of collaborators is expected to be high, the selected tool should be one that raises fewer conflicts.

When seeking performance, repository management or maintainability features, one should probably select a non-XML-aware tool like Normalised Git. If, one is seeking for XML specific features then XML-aware tools are likely the way to go, but it should be noted that some development work is still required to achieve parity with regards to the other features.

The research questions considered above (RQ1 to RQ3) only answer a portion of the required research to reach a more accurate conclusion on which approach is best. Aspects that were scoped out from this research such as different document type definitions, modularised documents, and the size of the content of a commit, should be considered as well.

7.2 Threats to validity and Ethical concerns

7.2.1 Conclusion Validity

Low Statistical Power: We did ten independent runs, with the variation on some parameters being high, increasing the number of independent runs would reduce the validity threat.

7.2.2 Internal Validity

Selection Bias: We've done the screening of different XML versioning tools, and might have left out relevant ones, for example when we excluded all non FOSS tools. On a similar aspect, we only interviwed a couple of different experts at the supporting company, increasing the possible selection bias.

We have set the results on the score sheet, and given eventual cognitive biases, there is the risk of having instrumented the results.

The implementations of the approaches under test have some adapter code that sets up the approaches and their tests. This has been designed and implemented by us and might not be optimal, therefore hindering the performance results.

7.2.3 Construct Validity

If the data set that was used does not reflect the average data set, the results may differ.

The consulted expert group consisted of two elements within the supporting company, it is a narrow sample with the same background. This may reduce the quality/quantity of the usage scenarios that were elicited in the initial part of the research.

Following that, our own interpretation of these scenarios that led to the final feature list might be faulty as well.

Feature Analysis Screening mode — The evaluation is based on our evaluation of third party information and some of the evaluation criteria are subjective.

7.2.4 External Validity

Results might not be generalisable to other kinds of XML documentation of those not being studied. Also, and perhaps more important, results might not be generalisable towards different scenarios that were not taken into account.

Only one computer and one operating system has been used in the performance tests, there is the possibility that the end results vary if these are run on a different environment.

The data set used for the tests was extracted from a single project, this implies that data may have consistent characteristics that may perform better in a specific approach.

The data set used on benchmarking were coming from a Git repository and that might create some bias towards Git, to balance that, we excluded from the set everything that was being rejected from Sirix (incorrect formatting, unresolved references, etc.).

7.2.5 Ethical concerns

It should be noted that this work was developed under the sponsorship of a supporting company and that the implementation of the XChronicler's [34] algorithm that was used was developed in a previous thesis work that was also sponsored by the same company.

Under this work we have tried to avoid our own biases or those of the supporting company. Having been kindly offered to use their anonymised client's data, we used technically similar open data instead in order to facilitate the replication of the research.

8

Conclusion

This thesis work evaluated and compared two different approaches regarding the versioning of XML, one being XML-aware and the other targeting plain text. Its outcome is not an absolute result, and no strong conclusions were able to be extracted from it. The results show that performance wise, the non-XML-aware approach is faster and consumes fewer resources, while meeting some basic needs regarding the problem.

However, and as pinpointed by the consulted experts, when adding more complex operations such as detection of move of elements, these tools fail considerably to handle them adequately.

In the following section we present the conclusions regarding the research questions.

8.1 Research Questions

8.1.1 RQ1: What features are required for a version control system to have in a documentation context?

Our research shows that there are many specific aspects to take into account when managing changes in XML within documentation context. We identified 6 different categories of features to take into account when choosing a version control system for XML, these are: Versioning; Repository Management; Merge/Conflict Resolution; XML Specific; CMS Features; and Project Maintainability.

8.1.2 RQ2: Of the previously identified features, which ones do XML-aware and non-XML-aware tools have?

From the analysed tools, both the approaches have good versioning and conflict resolution features, but there's a substantial difference among their strengths regarding the

other studied features. XML-aware tools are mostly strong, as expected, with XML related features, whereas the non-XML-aware approaches studied showed a major strength in repository management.

It should also be noted that the non-XML-aware approaches score very high in Project Maintainability, this is mostly due to the excellent status and reputation of the Git project that backs it up.

8.1.3 RQ3: How much overhead does an XML-aware tool carry when compared to a non-XML-aware one?

A benchmark analysis was performed over the top scoring tools within each of the considered approaches, namely Normalised Git and Sirix. The results were quite favourable towards the first, with Normalised Git consuming a mere 1/238 of CPU, 1/8 of RAM, and 1/132 of storage, while performing in 1/6 of the time that Sirix took to complete the same tasks for the same input data.

These results confirm our expectations that XML-aware tools, due to their more complex nature, do bring an overhead. Despite the previous statement, we cannot claim the generalisation of this major difference as we did not consider performance in the phases that led to the selected tools for benchmarking and one of the tools is in a highly mature state whether the other is not. Hence, we can only conclude that there is plenty of overhead added with **this** implementation of an XML-aware tool.

8.1.4 RQ0: What is a good approach to versioning an XML document?

To answer what is a good approach to versioning an XML document, this research focused on XML-aware and non-XML-aware approaches.

The overall results indicate that XML-aware tools are a good choice when in the need of tracking moves of elements within a document, track changes on a node, retrieve the node history, or if the amount of predicted conflicts during the lifetime of a document is to be high and the performance is not a issue.

Conversely the non-XML-aware tools are a good choice when in need of a trusted and more mature tool, with a higher level of documentation, or, if performance is a major concern.

In conclusion, both approaches fit to the versioning XML problem, which one to use will depend on the characteristics of the documentation project.

8.2 Future work

There are many possible ways to proceed with this work, from evaluating different versioning tools and approaches, to the use of different methodologies for the evaluation.

The elicitation of features with a different group of experts would be quite important to provide validation to the first part of this work.

Regarding different approaches to be evaluated, Operational Transformation might be an interesting one that seeks the support of collaborative authoring through different change synchronisation methods.

While this work was undergoing, many changes have happened within the tools and approaches under study, e.g. Git got a new major version, Sirix has been in continuous development, XQuery standard got updated, therefore an updated study on the same tools is not to be excluded as well.

On the benchmark analysis, the tests performed include only writing operations (commits). It might be of interest for future work to also benchmark reading operations (checkout of latest revision and also revisions in random places in the repository history) on the same tools as these are the most common operations performed in a documentation project.

Bibliography

[1] A. R. Hevner, S. T. March, J. Park, S. Ram, Design science in information systems research, MIS quarterly 28 (1) (2004) 75–105.

[2] S. Rönnau, U. M. Borghoff, Collaborative xml document versioning, in: Computer Aided Systems Theory-EUROCAST 2009, Springer, 2009, pp. 930–937.

[3] I. C. Society, Guide to the Software Engineering Body of Knowledge - Version 3.0 - SWEBOK, 2014th Edition, IEEE Press, 2014.
URL http://www.computer.org/portal/web/swebok/

[4] J. Paoli, T. Bray, M. Sperberg-McQueen, XML 1.0 recommendation, W3C recommendation, W3C (Feb. 1998).
URL http://www.w3.org/TR/1998/REC-xml-19980210

[5] E. Maler, F. Yergeau, T. Bray, M. Sperberg-McQueen, J. Paoli, Extensible markup language (XML) 1.0 (fifth edition), W3C recommendation, W3C (Nov. 2008).
URL http://www.w3.org/TR/2008/REC-xml-20081126/

[6] xmlfiles.com, Xml attributes — xml files (2000).
URL www.xmlfiles.com/xml/xml_attributes.asp

[7] E. R. Harold, XML Bible (Gold Edition), Hungry Minds, 2001.

[8] J. Boyer, Canonical XML version 1.0, W3C recommendation, W3C (Mar. 2001).
URL http://www.w3.org/TR/2001/REC-xml-c14n-20010315

[9] G. Marcy, J. Boyer, Canonical XML version 1.1, W3C recommendation, W3C (May 2008).
URL http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/

[10] W3C, W3c xhtml2 working group home page (2010).
URL http://www.w3.org/MarkUp/

[11] J. Simeon, M. Fernandez, J. Robie, D. Chamberlin, D. Florescu, S. Boag, XQuery 1.0: An XML query language (second edition), W3C recommendation, W3C (Dec. 2010).
URL `http://www.w3.org/TR/2010/REC-xquery-20101214/`

[12] J. Snelson, M. Dyck, J. Robie, D. Chamberlin, XQuery 3.0: An XML query language, W3C recommendation, W3C (Apr. 2014).
URL `http://www.w3.org/TR/2014/REC-xquery-30-20140408/`

[13] Wikipedia, FLWOR — Wikipedia, The Free Encyclopedia, [Online; accessed 4-June-2014] (2013).
URL `http://en.wikipedia.org/w/index.php?title=FLWOR&oldid=572903989`

[14] S. DeRose, J. Clark, XML path language (XPath) version 1.0, W3C recommendation, W3C (Nov. 1999).
URL `http://www.w3.org/TR/1999/REC-xpath-19991116`

[15] J. Melton, J. Simeon, D. Florescu, M. Dyck, D. Chamberlin, J. Robie, XQuery update facility 1.0, W3C recommendation, W3C (Mar. 2011).
URL `http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/`

[16] Ronald Bourret, rpbourret.com - xml database products (2014).
URL `http://www.rpbourret.com/xml/XMLDatabaseProds.htm`

[17] I. Corporation, DB2 Version 10.1 for Linux, UNIX, and Windows, [Online; accessed 14-December-2014] (September 2014).
URL `http://www-01.ibm.com/support/knowledgecenter/SSEPGG_10.1.0/com.ibm.db2.luw.xml.doc/doc/c0022308.html/`

[18] Microsoft, Using XML in SQL Server, [Online; accessed 14-December-2014] (September 2014).
URL `http://msdn.microsoft.com/en-us/library/ms190936(v=sql.90).aspx`

[19] Oracle, XMLType Operations, [Online; accessed 14-December-2014] (August 2005).
URL `http://docs.oracle.com/cd/B19306_01/appdev.102/b14259/xdb04cre.htm`

[20] postgresql, 8.13. XML Type, [Online; accessed 14-December-2014] (August 2013).
URL `http://www.postgresql.org/docs/9.0/static/datatype-xml.html`

[21] baseX, BaseX. The XML Database., [Online; accessed 14-December-2014] (August 2014).
URL `http://basex.org/`

[22] eXist Solutions, exist — vitamins for your applications (2014).
URL `http://exist-db.org/`

[23] marklogic, New generation big data requires a new generation database., [Online; accessed 14-December-2014] (December 2014).
URL `http://www.marklogic.com/`

[24] sedna, Native XML Database system, [Online; accessed 14-December-2014] (December 2014).
URL `http://www.sedna.org/`

[25] J. Lichtenberger, SirixDB github (2014).
URL `https://github.com/sirixdb/sirix`

[26] E. W. Myers, Ano (nd) difference algorithm and its variations, Algorithmica 1 (1-4) (1986) 251–266.

[27] S. Rönnau, Efficient change management of xml documents, Ph.D. thesis, Universitätsbibliothek der Universität der Bundeswehr (2010).
URL `http://athene-forschung.unibw.de/doc/88983/88983.pdf`

[28] A. Mouat, Xml diff and patch utilities, CS4 Dissertation, Heriot-Watt University, Edinburgh, Scotland, Senior Project.

[29] S. Rönnau, U. M. Borghoff, Xcc: change control of xml documents, Computer Science-Research and Development 27 (2) (2012) 95–111.

[30] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, Xml document versioning, ACM SIGMOD RECORD 30 (3) (2001) 46–53.
URL `http://www.sigmod.org/publications/sigmod-record/0109/SPECIAL/zaniolo7.pdf`

[31] F. Wang, C. Zaniolo, Temporal queries and version management in xml–based document archives, Data & Knowledge Engineering 65 (2) (2008) 304–324.

[32] G. Fourny, D. Florescu, D. Kossmann, M. Zaharioudakis, D. Kossmann, D. Kossmann, A time machine for xml, Tech. rep., ETH, Department of Computer Science (2011).
URL `http://e-collection.library.ethz.ch/eserv/eth:5661/eth-5661-01.pdf`

[33] G. Fourny, D. Florescu, D. Kossmann, M. Zaharioudakis, A time machine for xml: Pul composition, in: Kosek [53], pp. 233–242.
URL `http://archive.xmlprague.cz/2010/files/XMLPrague_2010_Proceedings.pdf`

[34] H. Svallfors, Efficient Temporal Queries in an XML-Based Content Management System, Master's thesis, Chalmers University of Technology, Gothenburg, Sweden (2013).
URL `http://studentarbeten.chalmers.se/publication/193724-efficient-temporal-queries-in-an-xml-based-content-management-system`

[35] J. Lichtenberger, A visual analytics approach for comparing tree-structures, Master thesis, University of Konstanz — Department of Computer and Information Science, Konstanz, Germany (2012).
URL http://kops.ub.uni-konstanz.de/handle/urn:nbn:de:bsz:352-210419

[36] T. Lindholm, J. Kangasharju, S. Tarkoma, Fast and simple xml tree differencing by sequence alignment, in: Proceedings of the 2006 ACM symposium on Document engineering, ACM, 2006, pp. 75–84.

[37] R. La Fontaine, Merging xml files: a new approach providing intelligent merge of xml data sets, in: Xml Europe, 2002, pp. 03–03.

[38] K. Komvoteas, Xml diff and patch tool, Ms in distributed multimedia and information systems dissertation, Heriot-Watt University, Edinburgh, Scotland (2003).

[39] Y. Wang, D. J. DeWitt, J.-Y. Cai, X-diff: An effective change detection algorithm for xml documents, in: Data Engineering, 2003. Proceedings. 19th International Conference on, IEEE, 2003, pp. 519–530.

[40] S. Rönnau, G. Philipp, U. M. Borghoff, Efficient change control of xml documents, in: Proceedings of the 9th ACM symposium on Document engineering, ACM, 2009, pp. 3–12.

[41] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, D. Zhang, Storing and querying multiversion xml documents using durable node numbers, in: Web Information Systems Engineering, 2001. Proceedings of the Second International Conference on, Vol. 1, IEEE, 2001, pp. 232–241.
URL http://www.cs.ucla.edu/~zaniolo/papers/wise01.pdf

[42] S. Graf, Treetank, a native xml storage, Tech. rep., University of Konstanz (2009).

[43] S. Graf, M. Kramis, M. Waldvogel, Treetank, designing a versioned xml storage.
URL http://kops.ub.uni-konstanz.de/handle/urn:nbn:de:bsz:352-opus-126912

[44] Sebastian Graf, Treetank — secure treebased storage (2014).
URL http://treetank.org/

[45] S. Graf, Flexible secure cloud storage, Doctor of engineering dissertation, University of Konstanz — Department of Computer and Information Science, Konstanz, Germany (2014).
URL http://kops.ub.uni-konstanz.de/handle/urn:nbn:de:bsz:352-272505

[46] K. Peffers, T. Tuunanen, M. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, J. Manage. Inf. Syst. 24 (3) (2007) 45–77. doi:10.2753/MIS0742-1222240302.
URL http://www.jmis-web.org/articles/v24_n3_p45/

[47] B. Kitchenham, S. Linkman, D. Law, DESMET: a methodology for evaluating software engineering methods and tools, Computing & Control Engineering Journal 8 (3) (1997) 120–126.

[48] B. Kitchenham, Evaluating software engineering methods and tools - part 2: Selecting an appropriate evaluation method - technical criteria, SIGSOFT Softw. Eng. Notes 21 (2) (1996) 11–15. doi:10.1145/227531.227533.
URL http://doi.acm.org/10.1145/227531.227533

[49] J. Kosek (Ed.), Conference Proceedings of the XML Prague, 2014.
URL http://archive.xmlprague.cz/2014/files/xmlprague-2014-proceedings.pdf

[50] J. Kosek (Ed.), Conference Proceedings of the XML Prague, 2013.
URL http://archive.xmlprague.cz/2013/files/xmlprague-2013-proceedings.pdf

[51] J. Kosek (Ed.), Conference Proceedings of the XML Prague, 2012.
URL http://archive.xmlprague.cz/2012/files/xmlprague-2012-proceedings.pdf

[52] J. Kosek (Ed.), Conference Proceedings of the XML Prague, 2011.
URL http://archive.xmlprague.cz/2011/files/xmlprague-2011-proceedings.pdf

[53] J. Kosek (Ed.), Conference Proceedings of the XML Prague, 2010.
URL http://archive.xmlprague.cz/2010/files/XMLPrague_2010_Proceedings.pdf

[54] L. E. Mendoza, M. Pérez, A. Grimán, Critical success factors for managing systems integration, Information Systems Management 23 (2) (2006) 56–75.

[55] B. Kitchenham, DESMET: A method for evaluating Software Engineering methods and tools, Technical Report TR96-09, Department of Computer Science — University of Keele, University of Keele, Keele, Staffordshire, ST5 5BG, U.K., iSSN:1353-7776 (1996).

[56] I. Black Duck Software, SirixDB github (2015).
URL https://github.com/blackducksw/ohcount

[57] OpenStack, Openstack manuals, https://github.com/openstack/openstack-manuals, hash of last commit used: 4e94b84d7e125761d36c1f900b09ccd33aa93d0a (Aug. 2014).

[58] Play Framework, Java play framwork (2015).
URL https://www.playframework.com/

[59] Eclipse foundation, Jgit (2015).
URL `https://eclipse.org/jgit/`

[60] xmllint, xmllint (2015).
URL `http://xmlsoft.org/xmllint.html`

[61] Jim / James C. Warner, top(1) - linux man page4 (2015).
URL `http://linux.die.net/man/1/top`

[62] Apache Foundatuion, Apache license, version 2.0, january 2004 (2015).
URL `https://www.apache.org/licenses/LICENSE-2.0.html`

[63] Blackduck, Discover, Track and Compare Open Source, [Online; accessed 7-February-2015] (February 2015).
URL `https://www.openhub.net`

[64] kernel.org, git/git @ github (2015).
URL `https://github.com/git/git`

[65] selenic, mercurial from selenic (2015).
URL `http://mercurial.selenic.com/`

[66] Google Developers, Minify resources (html, css, and javascript) - pagespeed insights — google developers (2015).
URL `https://developers.google.com/speed/docs/insights/MinifyResources`

[67] B. W. Boehm, et al., Software engineering economics, Vol. 197, Prentice-hall Englewood Cliffs (NJ), 1981.

Glossary

AJAX asynchronous JavaScript and XML(AJAX) is a group of techniques that is used in web development for building asynchronous Web applications. 34

Common API developed for this evaluation. Common API is the interface layer that lies on top of the different adapters for the versioning tools.. 31, 33, 34

delta file The same as a diff file, it is a representation of the differences between two versions of a document. 15

DESMET DESMET – a methodology for evaluating software engineering methods and tools, proposed by Kitchenham[55]. It identifies nine methods of evaluation and a set of criteria to help evaluators select an appropriate method.. 21, 22, 25

diff Diff is used in jargon as a verb for calculating any difference between two different documents. It is also referred to a file that contains the calculated changes required to transform a document A into a document B. 12, 34, 36

document type definition A document type definition defines the legal building blocks of an XML document. 3, 56

eXist High-performance native XML database engine and all-in-one solution for application building.[22]. 8, 12, 26, 35, 42, 44–49, 54

Git Git is a distributed revision control and source code management system with an emphasis on speed[64]. ii, 2, 24, 26, 30, 34, 40, 42, 44–50, 54–57, 83

Java Play Java Play is a web framework for developing web applications[58]. 34

JavaScript also known as ECMAScript, is a dynamic programming language for the web understood by all major web browsers, developed by Brendan Eich in 1995.. 34

JSON JavaScript Object Notation(JSON) is a lightweight data-interchange format. Designed for easy reading by humans and computers. 34

mercurial Mercurial defines itself as a free, distributed source control management tool. It efficiently handles projects of any size and offers an easy and intuitive interface [65]. 2

Minification Minification is the process of removing unnecessary or redundant data without affecting how the resource is processed[66].. 12

non-XML-aware a versioning tool that is not aware if it storage XML or plain text. 2, 3, 18, 30, 54–56, 59, 60

Normalised Git An application developed for this thesis work, that pre-processes the input (normalisation) before committing it to Git. 25, 26, 33, 35, 40, 42, 44–50, 54–56, 60, 81

Operational Transformation Operational transformation is a standard for a wide rage of collaboration functionalities. 61

patch a series of instructions that when added to a original document A, produce a document B. 30, 36, 39, 47, 73, 76

Pending Update List A pending update list is an unordered collection of update primitives, which represent node state changes that have not yet been applied. 11, 30, 34, 36

revision control system A software implementation of revision control that automates the storing, retrieval, logging, identification, and merging of revisions. 13, 17

Sirix Versioned XML database in Java, with added temporal XQuery[25]. ii, vii, 12, 16, 17, 25, 26, 30, 34, 36, 40, 42, 44–50, 54, 56, 57, 60, 61, 82, 83

Source Code Control System An early revision control system, geared toward program source code and other text files. 17

Test API developed for this evaluation. It is the interface that is used to retrieve a test and store its results.. 31, 33, 34

Test GUI developed for this evaluation. The Test GUI is the interface build for running the tests.. 31, 33

V-Document An XML document that describes another document's revision history. 15, 35

version control system A system that keep track of the changes to documents, it should address issues as branching, baselines, change-list, etc.. 3, 59

XChronicler Differencing and versioning tool that produces a V-Document[34]. ii, 15, 25, 26, 30, 35, 40, 42, 44–49, 54, 58

XML-aware a versioning tool that can understand XML structure. ii, 3, 11, 18, 29, 30, 55, 56, 59, 60

XPath XPath is a query language mainly used to select/address parts of an XML document. 8, 9

XPointer XPointer, or XML Pointer Language, is a language that allows the addressing of specific fragments of an XML document. 8

XQuery XQuery is a query language for XML. i, viii, 7–9, 11, 17, 30, 35, 36, 61

XSLT XSLT, or Extensible Stylesheet Language Transformations, is a transformation language for XML documents that transforms a document into another. 8

Acronyms

C14N Canonicalization. 7, 30, 35

CSF Critical Success Factor. 23

CVS Concurrent Version System. 2

DMS Document Management System. 23

FOSS Free and Open-Source Software. 3, 57

LPE Location Path Expression. 9

ODF Open Document Format. 3

SQL Standard Query Language. 7, 8

SVN Subversion. 2, 24

W3C World Wide Web Consortium. 5, 7–9, 30

XML eXtensible Markup Language. 1–3, 5–8, 11, 17, 18, 30, 35, 41, 53, 55, 56, 59, 60

XQUF XQuery Update Facility. 9

A

User Stories

Table A.1: User Stories

User Stories		
id	**title**	**description**
US-01	Parallel edit without conflicts	More than one person editing the same file in different parts at the same time
US-02	Parallel edit with conflicts and manual resolution	More than one person editing the same file in the same part at the same time
US-03	Parallel edit with conflicts and manual resolution	More than one person editing the same file in the same part at the same time
US-04	Parallel remove file	More than one person touching a file, one removes it while the other edits it
US-05	Commit to wrong branch	When you figure out that you have changes done on the wrong product line, and don't want to redo the changes
US-06	Serial simple edit file	More than one person editing the same file in the same part non-concurrently
US-07	Simple parallel edit on different branches	More than one person editing the same file in the same part at the same time but in different branches
US-08	Merge different branches	Merge changes that have been done in a separate branch ant to get all change from another branch

User Stories (cont.)		
id	**title**	**description**
US-09	Simple revert file to the immediate previous version	After committing a change in a file I want to revert to the previous version
US-10	Simple revert file to a version of choice	I want to revert a file to a previous version of my choice
US-11	Simple revert file to a version of choice with conflict	A user reverts file to a previous version before another user commits.
US-12	Remove element with conflicts	A user removes an element while another edits it.
US-13	Rename element with conflicts	Two users edit the same element at the same time.
US-14	Edit text with conflicts	Two users edit the same text within an element.
US-15	Remove text with conflict	Two users edit the same text within an element, one removes it before the other one changes its content.
US-16	Rename element and edit text	One user changes element's name while the other edits its content.
US-17	Merge a branch with a specific patch	merge a branch with a specific patch from another branch
US-18	Merging equal commits	Two users edit a file and both changes are equal.
US-19	Move element within a document	A user moves an element from one place to another.
US-20	Merge an change from repository to a Exported file	A user want to have his changes in repository reflected on the similar Exported xml
US-21	Merge a change on a Exported file to repository	A user want to have his changes in similar Exported XML reflected on the repository
US-22	Merge a part of a Exported file with an include file	A user wants to merge a part of Exported file (part of a whole without references/includes) with an included file (part)
US-23	Persistent reuse of part of a file in a different one	A user wants to be able to reuse part of a file in a different one and persist that connection
US-24	A user Exports a document	A user intends to extract a flattened document from the system

User Stories (cont.)		
id	**title**	**description**
US-25	A user Exports part of a document	A user intends to extract a flattened section of a document from the system
US-26	Different encoding	The use of different applications and systems, might save the document with different character encoding
US-27	Different indentation	The use of different applications and systems, might save the document with different indentation (non-significant white-space)
US-28	Change order of attribute	A user don't need to worry about the order of the attributes
US-29	Add a file	A user wants a file to be versioned
US-30	Remove a file	A user wants to remove a file from the versioning system
US-31	Edit a file	A user wants to change the contents of a file
US-32	Fork repository	A user wants to fork a repository in order to be able to make changes without impacting the original repository
US-33	Fork a part of a document	A user wants to fork a part of a document in order to be able to make changes without impacting the original part
US-34	Fork a document	A user wants to fork a document in order to be able to make changes without impacting the original document
US-35	checkout a revision of a document	A user intends to checkout a specific revision of a document
US-36	Commit to the repository	A user intends to save changes into the versioning system
US-37	Create the repository	A user intends to initialise a repository
US-38	Get list of changes	A user intends to understand the recent history of changes of a document

B

Feature list and Metrics

Table B.1: Versioning Features

Versioning Features				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
V-01	Fulfilment of tracking of file changes	5 - fulfilled 1 - not fulfilled	1	5
V-02	Fulfilment of reversion of a file to a previous version of choice	5 - fulfilled 1 - not fulfilled	1	5

Table B.2: Repository Management Features

Repository Management Features				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
RM-01	Existence of different branches of the same repository	5 - exists 1 - does not exist	1	5
RM-02	Ability to tag a specific revision of a repository	5 - exists 1 - does not exist	1	5

Repository Management Features (cont.)				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
RM-03	Ability to merge different branches of the same repository	5 - exists 1 - does not exist	1	5
RM-04	Ability to remove/untrack a file on the repository	5 - exists 1 - does not exist	1	5
RM-05	Ability to amend a commit	5 - exists 1 - does not exist	1	5
RM-06	Ability to rename a file	5 - exists 1 - does not exist	1	5
RM-07	Ability to fork a document	5 - exists 1 - does not exist	1	5
RM-08	Ability to handle different users	5 - exists 1 - does not exist	1	5

Table B.3: Merging / Conflict resolution Features

Merging / Conflict resolution Features				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
MCR-01	Ability to handle conflicts	5 - exists 1 - does not exist	1	5
MCR-02	Ability to merge different patches	5 - exists 1 - does not exist	1	5
MCR-03	Ability to interactively merge hunks of a file	5 - exists 1 - does not exist	1	5
MCR-04	Handling of non significant white-space	5 - exists 1 - does not exist	1	5

Table B.4: Project Maintainability Features

Project Maintainability Features				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
PM-01	Level of Popularity of the project (*where x = github stars*)	5: $\{x > 10.000\}$ 4: $\{x \leq 10.000\}$ 3: $\{x \leq 1.000\}$ 2: $\{x \leq 100\}$ 1: $\{x \leq 10\}$	1	5
PM-02	Level of Activity of the project (*where x = commits to SCM in 2014*)	5: $\{x > 10.000\}$ 4: $\{x \leq 10.000\}$ 3: $\{x \leq 1.000\}$ 2: $\{x \leq 100\}$ 1: $\{x \leq 10\}$	1	5
PM-03	Level of documentation of the project *(where x = Book hits* $\times$ 100 + +*Search Engine hits* $\times$ 0.01 + +*Expert Community Q&A hits* $\times$ 1 + +*Source Code lines of comment* $\times$ 0.01 + +*Official Documentation lines of text* $\times$ 0.1	5: $\{ x > 100.000\}$ 4: $\{ x \leq 100.000\}$ 3: $\{ x \leq 10.000\}$ 2: $\{ x \leq 1.000\}$ 1: $\{ x \leq 100\}$	1	5

Table B.5: XML Specific Features

XML Specific Features				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
XML-01	Handling insertion of child (*where x = min(tests)*)	5: x :Possible and can not create unintended conflicts 3: x :Possible but can create unintended conflicts 1: x :Not Possible	1	5

XML Specific Features (cont.)				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
XML-02	Handling insertion of sibling	5: x :Possible and can not create unintended conflicts 3: x :Possible but can create unintended conflicts 1: x :Not Possible	1	5
XML-03	Handling insertion of parent	5: x :Possible and can not create unintended conflicts 3: x :Possible but can create unintended conflicts 1: x :Not Possible	1	5
XML-04	Handling of element rename	5: x :Possible and can not create unintended conflicts 3: x :Possible but can create unintended conflicts 1: x :Not Possible	1	5
XML-05	Handling of removal of elements	5: x :Possible and can not create unintended conflicts 3: x :Possible but can create unintended conflicts 1: x :Not Possible	1	5
XML-06	Handling of move of an element	5: x :detects move operations 3: x :detects removal and insertion operations 1: x :fails to handle	1	5

XML Specific Features (cont.)				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
XML-07	Handling of attributes insertion	5: x :Possible and can not create unintended conflicts 3: x :Possible but can create unintended conflicts 1: x :Not Possible	1	5
XML-08	Handling of attributes removal	5: x :Possible and can not create unintended conflicts 3: x :Possible but can create unintended conflicts 1: x :Not Possible	1	5
XML-09	Ignores the change of order of an attribute	5: Yes 1: No	1	5
XML-10	Ability to enforce well-formed xml on the repository	5 - exists 1 - does not exist	1	5
XML-11	Ability to track changes on a node	5 - exists 1 - does not exist	1	5
XML-12	Ability to retrieve the history of a specific node	5 - exists 1 - does not exist	1	5

Table B.6: CMS Specific features

CMS Specific features				
id	**Metric Name**	**Formulation**	**Lower Value**	**Higher Value**
CMS-01	Ability to retrieve list of changes (log). expected features : {Author, Message, Timestamp, Reference to commit} $\left(where\ x = \frac{features\ complied}{total\ expected\ features} \times 5\right)$	5: $\{x > 4\}$ 4: $\{4 > x \geq 3\}$ 3: $\{3 > x \geq 2\}$ 2: $\{2 > x \geq 1\}$ 1: $\{1 > x \geq 0\}$	1	5
CMS-02	Ability to retrieve list of modified elements in the current working copy	5 - exists 1 - does not exist	1	5
CMS-03	Ability to flatten a modular document	5 - exists 1 - does not exist	1	5
CMS-04	Ability to reuse parts of a file in others	5 - exists 1 - does not exist	1	5

C

Benchmark Results

Table C.1: Benchmark results: Normalised Git

Benchmark results: Normalised Git				
Test Number	**CPU (%)**	**Hard-Drive Size (B)**	**Memory (% of 8 GB)**	**Time (ns)**
1	1.04	7397.24	10.09	120005257.0
2	1.11	7397.20	6.54	120854531.4
3	0.93	7396.39	3.37	120217476.3
4	1.19	7395.89	6.54	120341663.3
5	1.34	7396.57	6.55	120627771.1
6	1.23	7396.12	6.61	120283701.2
7	1.63	7396.13	6.65	122876457.9
8	1.26	7396.84	6.55	120818115.7
9	1.45	7396.37	6.59	120860879.5
10	1.59	7396.78	6.64	121801376.2
Average	1.24	7396.53	6.61	120765094.8
Median	1.23	7396.39	6.55	120627771.1
Standard Deviation	0.21	0.48	1.68	850365.4

Table C.2: Benchmark results: Sirix

Benchmark results: Sirix				
Test Number	**CPU (%)**	**Hard-Drive Size (B)**	**Memory (% of 8 GB)**	**Time (ns)**
1	44.60	974542.84	9.35	596174011.8
2	44.88	974191.26	9.78	612656860.7
3	43.96	972715.81	9.02	553498619.7
4	51.95	974390.84	9.38	998883915.9
5	50.00	973564.06	8.19	903283391.2
6	52.27	973991.31	9.27	930158363.5
7	54.15	971809.19	9.32	719727839.0
8	56.10	973396.33	9.57	709110878.4
9	56.67	973921.50	9.58	691684147.6
10	48.07	974327.50	9.28	484740026.0
Average	50.27	973685.06	9.27	719991805.4
Median	50.98	973956.41	9.34	700397513.0
Standard Deviation	4.75	856.93	0.43	172040499.6

D

Project Metrics Analysis

Table D.1: Project Metrics Analysis

Project Metrics Analysis[63]		
Project name	**Git**	**Sirix**
Number of commits	38,464	553
Number of contributors	1,289	6
Lines of code	342,067	74,650
Main language	C	Java
Number of comments	average	above average
Team size	very large	average
COCOMO model[67]	92 years	19 years
First commit	April, 2005	June, 2012

E

Pre-processing steps of benchmark data

Listing E.1: source-code of the pre-processing script used on the input data for benchmark

```
import os, subprocess, shutil, sys, re

rootdir = '.'
folderUrl = "Uri/To/Folder";

for subdir, dirs, files in os.walk(rootdir):
  for file in files:
    counter = 0
    shutil.rmtree(folderUrl + file + "mapp", ignore_errors = True)
    os.mkdir( folderUrl + file + "mapp", 0755 );
    print "git log on: "+file
    ps = subprocess.Popen(('git', 'log', '--stat', file), stdout =
        subprocess.PIPE)
    output = subprocess.check_output(('grep', 'commit'), stdin =
        ps.stdout)
    ps.wait()
    for item in reversed(output.split()):
      if item != "commit":
        print "checkout: " + item
        counter += 1
        gitCheckout = subprocess.Popen(('git', 'checkout', item), stdout
            = subprocess.PIPE)
        gitCheckout.wait()
```

```
21          with open (file, "r") as myfile:
22            data = myfile.read()
23
24          data = re.sub(r"\&ndash;", "", data)
25          data = re.sub(r"\&mdash;", "", data)
26          data = re.sub(r"\ ", "", data)
27          data = re.sub(r"\&lt;", "", data)
28
29          data = re.sub(r"<!--.*-->", "", data)
30          data = re.sub(r"<xi:.*/>", "", data)
31          data = re.sub(r"<\?.*\?>", "", data)
32
33          #remove namespaces
34          data = re.sub(r"xlink:", "", data)
35          data = re.sub(r"xml:", "", data)
36
37          def removeNewLine(matchobj):
38            return re.sub(r'\n', " ", matchobj.group(0))
39          data = re.sub(r'\<((.|\n)*?)\>', removeNewLine,data)
40
41          with open(folderUrl + file + "mapp" + '/' +
42          tr(counter).zfill(2), "w") as text_file:
43            text_file.write("%s" % data)
44
45
46      gitCheckout = subprocess.Popen(('git', 'checkout', 'master'),
            stdout=subprocess.PIPE)
47      gitCheckout.wait()
```

F

Environment Specification

The test environment used for the performance tests was the following:

Laptop brand/model Asus U36JC

CPU Intel®Core™ i5 CPU M 480 2.67GHz x 4

Hard-drive Samsung SSD 840 EVO 250GB

RAM 8 GB of memory.

Operating System Ubuntu 14.04

Kernel Linux kernel 3.13.0-35-generic x86_64 GNU-Linux

File System EXT4

Libraries/Runtimes

- Java version 1.8.0_20
- Git version 1.9.1
- brackit-0.1.3-20140421.192832-4
- sirix-parent-0.1.3-20140520.173148-31

G

Test Script Example – Source Code

Listing G.1: source-code of the test script for a parallel editing

```
(function() {
  var userStoryUuid = "ParallelEditWithoutConflicts";
  var testName = "basic";
  console.log("Running " + userStoryUuid + ":" + testName);

  var testResult = false;
  var commonAPI = new CommonAPI();
  var userStory = getUserStory(userStoryUuid);
  var test = getTest(userStory,testName);
  var inputFolder = "/" + userStoryUuid + "/" + testName + "/input/";


  var fileName = testName,
  initialFile = test.input[0],
  initialFileMessage = "I HAVE 3 ELEMENTS: a,b,c",
  alicesFile = test.input[1],
  alicesFileMessage = "I HAVE 2 ELEMENTS: a,c",
  bobsFile = test.input[2],
  bobsFileMessage = "I HAVE 3 ELEMENTS: a,b,c AND d WITHIN a",
  expectedFile = test.input[3],
  expectedFileMessage = "I HAVE 2 ELEMENTS: a,c AND d WITHIN a",
  actualFile;
  //PRECONDITIONS
  // "Repository exists"
  commonAPI.initRepo().then(
```

```
26      function(data){
27        // "File exists" and "File Content exists"
28        return commonAPI.commitFile(fileName, inputFolder+initialFile,
              initialFileMessage, 0);
29      },
30      function(data){
31        console.log("Failed to initRepo");
32        reportIn(userStoryUuid, testName, "error");
33      }
34    ).then(
35      //SCENARIO
36      // "Alice checkouts repository" and "Alice edits file" -> alicesFile
37      // "Bob checkouts repository" and "Bob edits same file in different
            place" -> bobsFile
38      function(data) {
39        // "Alice commits changes"
40        return commonAPI.commitFile(fileName, inputFolder+alicesFile,
              alicesFileMessage, 0);
41      },
42      function(data){
43        console.log("Failed to commit initialFile");
44        reportIn(userStoryUuid, testName, "error");
45      }
46    ).then(
47      function(data){
48        // "Bob commits changes"
49        return commonAPI.commitFile(fileName, inputFolder+bobsFile,
              bobsFileMessage, 1);
50      },
51      function(data){
52        console.log("Failed to commit alicesFile");
53        reportIn(userStoryUuid, testName, "error");
54      }
55    ).then(
56    //OUTCOME
57    //testResult is true iff both commits are successful and Bob's commit
          includes Alice's changes as well.
58      function(data) {
59        return commonAPI.getHEAD();
60      },
61      function(data){
62        console.log("Failed to commit bobsFile");
63        reportIn(userStoryUuid,testName,"error");
64      }
65    ).then(
66      function(data){
67        //do comparison with expectedFile and actualFile
68        return compare(expected, actual);
69      },
70      function(data){
71        console.log("Failed to getHEAD");
```

```
72        reportIn(userStoryUuid, testName, "error");
73      }
74    ).then(
75      function(data){reportIn(userStoryUuid,testName,"success");},
76      function(data){
77        console.log("Failed to Compare");
78        reportIn(userStoryUuid, testName, "error");
79      }
80    );
81  }());
```