DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
71

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
71

SIIM KARUS

Maintainability of XML Transformations

TARTU UNIVERSITY PRESS

Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in informatics on May 13, 2011, by the Council of the Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu.

Supervisors:

Prof PhD Marlon Dumas
Institute of Computer Science
University of Tartu
Tartu, Estonia

Docent PhD Helle Hein
Institute of Computer Science
University of Tartu
Tartu, Estonia

Opponents:

Prof PhD Mehdi Jazayeri
Faculty of Informatics
University of Lugano
Lugano, Switzerland

Assistant professor PhD Martin Pinzger
Department of Software Technology
Delft University of Technology
Delft, The Netherlands

Commencement will take place on June 28, 2011, at 15.00 on Liivi 2–404

ISSN 1024–4212
ISBN 978–9949–19–695–1 (trükis)
ISBN 978–9949–19–696–8 (PDF)

Tartu Ülikooli Kirjastus
www.tyk.ee
Tellimus nr 325

CONTENTS

LIST OF ORIGINAL PUBLICATIONS

1. Karus, Siim; Gall, Harald. A Study of Language Usage Evolution in Open Source Software. *In: Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), May 21–22, 2011, Waikiki, Honolulu, HI, USA*. IEEE, 2011.
2. Karus, S.; Dumas, M. (2007). Enforcing Policies and Guidelines in Web Portals: A Case Study. International Workshops on Web Information Systems Engineering (WISE 2007 International Workshops). (Eds.)Mathias Weske, Mohand-Said Hacid, Claude Godart. Berlin: Springer, 2007, (Lecture Notes in Computer Science; 4832), 154–165.
3. Karus, Siim; Dumas, Marlon (2010). Designing Maintainable XML Transformations. *In: Software Maintenance and Reengineering, European Conference on: 14th European Conference on Software Maintenance and Reengineering, 15–18 March 2010, Madrid, Spain*. IEEE Computer Society, 2010, 137–145.
4. Karus, Siim; Dumas, Marlon. Predicting the maintainability of XSL transformations, Science of Computer Programming, In Press, Available online 31 December 2010, ISSN 0167-6423, DOI: 10.1016/j.scico.2010.12.006.
5. Karus, Siim; Dumas, Marlon. Predicting Code Churn from XML Metrics, Under Review.

ABSTRACT

Contemporary software systems often rely on the Extensible Markup Language (XML) to represent data internally and to interact with other systems and with end users. Concomitantly, the Extensible Stylesheet Language (XSL) is commonly employed to define transformations between the internal XML documents manipulated by a system and XML documents used for external interaction. These XSL transformations need to be updated whenever the underlying XML formats evolve, thereby raising a maintenance problem. In this setting, this dissertation undertakes to study the usage of XML and XSL in software systems and to devise models and guidelines to estimate and reduce the effort needed to maintain code in software projects that use XML.

The dissertation starts with an analysis of open source software repositories in view of unveiling usage patterns of XML files in open source projects. The analysis of software code repositories is followed by studies of maintainability and forward compatibility of XSL transformations (XSLT). Finally, the dissertation studies the influence of XML on the maintainability of software projects by means of exploratory data analysis.

The analysis of open source software repositories shows that XML files co-change with about 20% of files of other types (e.g. Java, C++), suggesting that significant relations may exist between the coding effort associated to XML code and the coding effort of the entire project. The analysis also revealed that both usage of XML in general and usage of XSLT in particular have been steadily increasing for the last decade, confirming the relevance of studying the evolution and maintenance of XML and XSL code in software projects.

The maintainability of XSL transformations is first approached using confirmatory data analysis by putting forward and testing a set of guidelines that encapsulate well-known principles of software maintainability – specifically forward compatibility. The study demonstrates a relation between the use of the guidelines and reduction in the amount of XSL code written and modified in the context of iterative software development projects. An exploratory approach based on machine learning and descriptive analysis on software repositories is then used to identify additional and more specific guidelines. The guidelines identified using the exploratory approach partially matched the guidelines tested using confirmatory data analysis. The models built using exploratory data analysis proved to be good estimators of XML-related code churn: the sum of added, removed and modified code during a period of time.

Finally, XML-related metrics are shown to influence long-term cumulative code churn of a project (including both XML and non-XML files) to a larger extent than some well-known object-oriented code metrics. Regarding long-term code churn estimation, the dissertation also reveals the existence of highly accurate estimation models based on organisational data present in source code repositories.

In conclusion, the dissertation advances the understanding of the factors that influence the maintainability of XML transformations, and more generally, the maintainability of software projects that use XML code. Based on the identified influences, guidelines for designing maintainable XML transformations are formulated. The predictive models developed in this study can also be useful in the planning process of software projects.

1. INTRODUCTION

Software is ubiquitous in today's society. Not only do we use Web-based software systems on a daily basis for communicating, reading news, socialising and shopping, but software also runs transportation and logistics systems, utilities, enterprise processes, business intelligence dashboards, refrigerators, cell phones, coffee machines and TV-sets. While traditionally much of this software has been closed source and commercial, open source software has emerged as a viable alternative paradigm for software production. The public nature of open source software makes it an appealing instrument for studying the practice of software development.

Given the ubiquity of software and the sheer size and complexity of some contemporary software systems, estimating and reducing the cost associated with software development and maintenance is a highly relevant problem. Studies on the costs associated with software development and maintenance give valuable insights for improving software development processes and aid in planning and evaluation of software development projects.

This dissertation tackles the task of understanding and managing the maintainability of software, with a focus on software that makes use of the Extensible Markup Language (XML) [1] and XML transformations. The dissertation looks at source code from different angles employing case studies of closed source software and applying data mining techniques on open source software in order to validate and refine indicators and guidelines of maintainability of XML transformations.

In the next sections, XML transformations and their role in software is explained along with technologies and concepts relating to transforming XML. This background information is followed with the introduction of the core questions and approaches used in the dissertation. Finally, the contributions and outline of the research are given to introduce the summaries of the contributing studies taken to answer the questions.

1.1. Background

This subsection gives a brief introduction to XML technologies and software maintenance concepts relevant to the dissertation, thereby setting the scene for the formulation of the problem statement.

1.1.1. Extensible Markup Language

Extensible Markup Language is an increasingly popular language for storing layout data. XML is a popular choice as an interoperable language for documents used for communicating with software services or components. XML is also the basis of widely spread languages like XHTML (The Extensible HyperText Markup Language [2]) and RSS (either RDF Site Summary [3] or Really Simple Syndication [4]) [5], which are focused on human users. It is also becoming more popular in software development as a definition language for program, presentation or storage logic. Examples of these definition languages include XAML (Extensible Application Markup Language [6]), Hibernate [7], and WS-BPEL (Web Services Business Process Execution Language [8]). XML is also used to define the structure of projects themselves as well as the software building processes (e.g. ant and Maven configurations). In summary, XML has become such a natural part of our digital environment that we often do not notice its presence.

Some of XML's popularity can be attributed to its simple design. XML document is composed of nodes structured as a tree. The nodes in the tree can be of only five different types: element, attribute, processing instruction, comment, and text. Each of these nodes has a value, the first three (cf. element, attribute, and processing instruction) also have a name, and element nodes can also have other nodes as children. Apart from a few restrictions on node names, this is all anyone wanting to use XML has to know to begin with. An example of XML file and its nodes tree structure layout is shown on Figure 1. This clear, simple and expressive structure is relatively easy to process by both humans and computers.

```xml
<?xml version="1.0" encoding="utf-8"?>
<links>
   <!-- The following two elements will
be selected by XPath expression
"//a[@class = 'web']" -->
   <a class="web"
href="http://vabavara.eu"/>
   <a class="web"
href="http://www.ut.ee"/>
   <!-- The next element will
additionally be selected by XPath
expression "//*[string-length(@class) =
3]" -->
   <a class="wmi"
href="\\Machine1\root\cimv2:Win32_Logic
alDisk.DeviceID='C:'">
      Drive C:
   </a>
   <!-- The next element will not be
selected by either XPath expression -->
   <a class="file"
href="file://D:/Documents/Intro.docx"/>
</links>
```

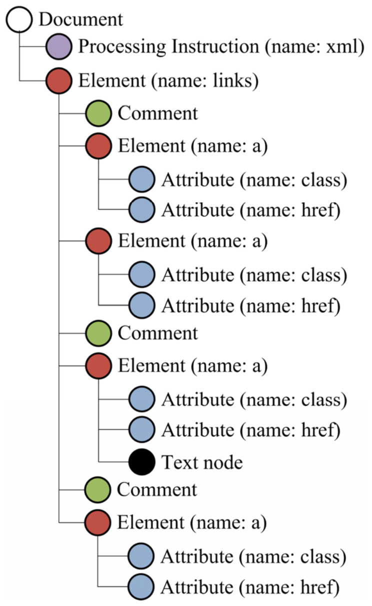


Figure 1. An example XML file and its tree representation

XML is the base for many languages (e.g. XHTML, RSS, WS-BPEL) and document formats such as HL7 (Health Level 7 [9]) – a widely used document format for data exchange in the domain of healthcare solutions – and ebXML (Electronic Business using Extensible Markup Language [10]) – a widely used document format for trading data exchange between companies. The existence of multiple XML languages and formats – an unavoidable requirement given the open (multidisciplinary) nature of the web – raises the need for developing transformations to and from XML document formats and languages. Recent trends such as Service-Oriented Architecture and Web 2.0 [11] – which rely on XML for data exchanges across software systems and sub-systems – have heightened the role of XML transformations in web information systems to unprecedented levels.

1.1.2. Extensible Stylesheet Language Transformations

Extensible Stylesheet Language Transformations (XSLT) [12] is often the preferred choice for specifying XML transformations due to its platform-independence and performance [13,14,15]. Even graphical editors used for defining XML transformations often generate XSLT code [16]. In many cases, XSLT is used in a supportive role as a language for accomplishing specific tasks and amounts to a rather small percentage of the total code base of a project [17]. However, considerable amounts of XSLT code can be found in projects that rely heavily on document transformation and data viewing. Reportedly, some applications in the financial domain contain libraries with tens of thousands lines of XSLT code[1]. This and similar examples show that the amount of XSLT is non-negligible and maintaining these transformations is an issue worth consideration.

XSLT belongs to the Extensible Stylesheet Language family (XSL) [18]. In addition to the transformations language XSLT, XSL also contains formatting language XSL Formatting Objects (XSL-FO) [19] and an expression language XPath (The XML Path Language) [20]. Unlike XSL-FO and XPath, which are stand-alone languages, XSLT is dependent on XPath for its need for addressing and selecting nodes in XML documents.

In our research we differentiate between simple and complex XPath expressions. Complex expressions are expressions that match a wide set of source structures, as opposed to expressions which only match very specific source structures (simple expressions). Simple expressions are all expressions that can be written without wildcards or function calls. For example "//a[@class = 'web']" (select all elements <a> which have attribute "class" value "web" – see Figure 1 for an example) is a simple expressions while "//*[string-length(@class) = 3]" (select all elements which have a value of length 3 for

[1] http://www.nycircuits.com/xml/xsl-consulting.html

attribute “class” – see Figure 1 for an example) is a complex expression. Complex expressions are more generic and we show that this genericity has an effect on maintainability [21,22].

1.1.3. Maintainability

Maintainability is the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [23]. In other words, maintainability is the ease with which maintenance can be accomplished. Maintenance of information system is defined as the effort of keeping information system operational and responsive to users after it is installed and in production [24]. This can also be applied to software systems and components (including source code).

According to [25] maintainability in (information) systems development is indicated by:

- The level of the methods and tools employed (compatibility condition);
- Software’s compliance with quality standards and the number of bugs (integrity condition);
- Modularisation and composition of software (simplicity condition);
- Motivation and apprehension of user requests for new functionality (usability condition);
- Amount of changes needed for addition of new modules (extensibility condition);
- Number of faults introduced by modifications (stability condition); and
- Experience of software maintainers with the information system (familiarity condition).

The conditions of maintainability relate to different sides of software development. For example, integrity, usability, and familiarity are conditions of software development’s organisational (administrative) side while simplicity, extensibility, and stability relate to software design (compatibility condition relates to both sides). Both of these sides are considered in this dissertation, however, a more suitable concept of forward compatibility is used as a basis for developing guidelines for achieving better maintainability of XML transformations.

1.1.4. Forward Compatibility

Forward compatibility is the system’s ability to gracefully accept input intended for later versions of itself. The idea behind forward compatibility is that evolutionary changes to software should be avoided and their extent minimised (the changes should be simple to implement). In 1999 Chris Armbruster outlined three properties of forward compatibility of software systems and components [26]: extensibility, abstraction, and componentisation.

Extensibility is the system's ability to be extended to handle methods, protocols, content, etc. that were not considered (e.g. did not exist) when the system was designed or re-designed. This can be achieved by updating the system whenever a change in the environment is made or by allowing the systems to negotiate the usage-level details needed to handle the extended data. This means creating systems that can either ignore the optional extended input or apply general non-feature-specific rules when handling new features.

Abstraction separates actual implementations from contracts. This can be used to simplify the contracts and allow gradual upgrading of the system (abstraction layer by layer). This also allows systems to evolve faster in response to changes in data representation.

Componentisation is similar to abstraction as it breaks up the system into components that can evolve independently. However, componentisation also deals with the choice of components used, allowing systems to choose the components used for input processing depending on the components the system is subscribed to.

By comparing the conditions of maintainability with the conditions of forward compatibility, it is clear that maintainability is a broader term as it also covers integrity, usability, stability, and familiarity. On the other hand, the most visible condition of extensibility is common to both. The condition of simplicity in maintainability maps to componentisation (modularisation condition) and abstraction (composition condition) in forward compatibility. As such, forward compatibility covers the part of maintainability that is associated with the design of software.

In this dissertation, we are concerned with formulating guidelines for achieving forward compatibility, and thus reducing the maintenance effort in the context of software projects that make use of XML and XML transformations. In order to test and quantify whether or not the proposed guidelines indeed lead to greater maintainability, a way of measuring maintainability has to be established. We could achieve this by assessing the "ease of performing maintenance tasks" but this property is largely subjective and does not lend itself to quantitative measurement. We can, however, quantify software maintenance effort and we can objectively assess the fulfilment of the conditions associated to maintainability or forward compatibility. In this way, we can determine if software projects that follow the proposed guidelines lead to systems that have a lower maintenance effort and/or to systems that fulfil the conditions associated with maintainability or forward compatibility.

Software maintenance effort is closely related to software development effort. The difference is that software development effort does generally also include effort made before the software is in production (released). This distinction is insignificant in case of managed software source code in open source software development as every commit to the source code management system is effectively a new release. Thus, development effort of open source software is indicative of maintenance effort of open source software.

1.1.5. Measures of Development Effort

There are many ways of estimating development effort [27,28,29,30,31,32]. Common measures used for effort estimation are code churn, code delta, and change rate (relative code churn) [33]. Code churn can also be successfully used to estimate the impact of a change in a project [34], making it preferable metric for our research.

Code churn is defined as the sum of code added, code removed, and code modified. This metric can be applied at different levels of granularity (i.e. method, class, line, file, module). As this research focuses on XML, it would not make sense to use granularity levels "method" or "class", however, line or file level churn can be measured on all text-based files. In our case only lines of code (LOC) churn was considered. LOC churn has been widely studied and has been shown to be useful in many estimation scenarios (e.g. defect estimation [35] or impact of change [34]).

In this dissertation two different timeframes are used for churn estimation: next revision churn (commit churn), and yearly cumulative code churn. As a commit to a source code repository signifies a completion of a task or change worth distributing, it can be considered an indicator of forward compatibility and maintainability of the code (less churn means less coding is needed to make a meaningful change). In contrast, yearly cumulative code churn (sum of revision code churns for all files in a project in a year) is more meaningful for coding effort estimation. Churn in a year long period is considered long-term churn and is well explored by previous studies of code churn [36,37].

It is important to note that better maintainability and lower maintenance (or development) effort do not always go hand-in-hand. Specifically, maintainability ignores maintenance effort in project planning and system design tasks. This difference is explored in [25]. The contributions of this thesis ought to be seen in light of the fact that we only measure coding effort, and not analysis, design or project management effort.

1.2. Problem Statement

The fact that XML is commonly a non-primal language in software projects [17], combined with the simplicity of XML, has caused the maintainability of XML and XML manipulations to be largely overlooked within the software engineering community. The research presented in this dissertation aims to fulfil that void by studying the role of XML and XML transformations in software projects and developing useful guidelines and models that can be used to improve the effectiveness of XML intensive software development.

As stated earlier, XML is used for numerous different purposes. This calls for a need to transform XML documents from one concrete representation to another. These transformations pose an additional development and maintenance effort.

This dissertation studies the usage of XML in software projects and looks for means to reduce the need and complexity of changes to XML transformations. The solutions discussed in this dissertation include offloading the task to other parties (working around the problem not solving it) and creating more maintainable transformations in terms of coding effort.

1.2.1. Specifying Requirements on XML Transformations

One way of reducing the development costs of XML transformations is to transfer the task of writing the transformations to third parties by exposing raw XML interfaces only. In other word, the XML data provider offloads the responsibility of writing XML transformations to third parties. The problem with this approach is that the third parties can generally do anything they wish with the data, in some cases against the interests of the data provider.

In order to allow the XML data providers to offload the maintenance effort of XML transformations, we used a language for specifying requirements on XML transformations, namely *xslt-req* [38]. This language extends XML Schema [39] – a standard language for specifying XML document structure. This language allows XML data providers to specify policies that a given XSL transformation must fulfil. The idea is that XML data providers will provide public-facing XML web services together with an associated schema extended with *xslt-req* rules. Third parties wishing to use this public-facing web service may provide their own XML transformations, which will be applied to the XML data produced by the web service, in order to produce (for example) HTML code. Importantly, the XML transformations registered by the third-parties in the XML data provider's system are automatically conformance-checked against the specified *xslt-req* rules. This automatic conformance checking makes the approach much more cost-effective than manual quality assurance of the XML transformations by the XML data provider [40]. This approach is akin to checking the validity of XML documents against XML Schemas, insofar as XSL transformation are additionally checked against *xslt-req* rules before they are accepted by the data provider's system.

In respect to specifying requirements on transformations, this dissertation investigates the following questions:

RQ1.1 What are the measurable benefits of using *xslt-req* as a language for specifying requirements of XML transformations in order to enforce the policies of XML data providers?

RQ1.2 What are the benefits of different techniques for enforcing guidelines on XML transformations?

1.2.2. Achieving Forward Compatibility of XML Transformations

It is not always possible to offload the maintenance effort of XML transformations. Also, by offloading the responsibility, we do not solve the problem in general – we only solve it for one interested party. Thus a closer look into reducing maintenance effort is needed. In this dissertation, we followed two tracks: 1) we tried to avoid the need for changes by improving the forward compatibility of XML transformations; and 2) we created models to estimate coding effort and to identify the drivers of maintainability effort.

With respect to the goal of achieving forward compatibility of XML transformations, the dissertation aims to answer the following questions:

RQ2.1 How is XML used in software projects?

RQ2.2 How do XML artefacts co-evolve with other artefacts in software projects?

RQ2.3 What guidelines can be given to XML transformation developers in order to promote the forward compatibility of XML transformations?

RQ2.4 Do the proposed forward compatibility guidelines affect the performance of the resulting XML transformations, and if so, to what extent?

1.2.3. Building Code Churn Estimation Models

As stated in Section 1.1.5, code churn is used to measure coding effort. Code churn can be associated with maintainability and be used in maintenance planning; hence, code churn estimation models can be used to improve maintainability of software source code.

Code churn estimation is well studied in the domain of procedural and object-oriented languages. Most of these models base their estimations on features like amount of code, number of classes, number of functions, complexity and other similar metrics related to the structure or program execution flow [41,36,42,43]. In this dissertation, metrics similar to those defined for object-oriented code are proposed for XML and XSLT code [44,22] and these metrics are used to investigate the role that XML and XSLT code play in relation to code churn. More specifically, the research presented in this dissertation aims to find answers to the following questions:

RQ3.1 Can we estimate XSL code churn of the next revision?

RQ3.2 What is the predictive power of metrics associated with XML for code churn estimation?

RQ3.3 Which factors influence code churn of the next revision of an XSL transformation?

RQ3.4 Which factors influence the total cumulative code churn of a project in the long term?

1.3. Approach

The problems outlined in the previous section require different approaches. When going in the direction of offloading development effort, we are met with problems related to the specification and enforcement of policies set by the provider of XML data. The enforcement of these policies can be difficult. In this dissertation we evaluate some mechanisms for enforcing policies on XSL transformations on a case study. Based on the experiences of the case study, development of guidelines for designing maintainable XML transformations is put forth.

The approach used in the dissertation for development of such guidelines is to adapt and refine existing maintainability-related guidelines in software development. As part of this approach a set of 4 guidelines is put to test on different XML transformations. The guidelines developed using this approach are complemented by rules found by code churn estimation models.

The approach used for building code churn estimation models makes use of machine learning algorithms and descriptive analysis techniques. The approach used in the estimation-related studies (publications 4 and 5) can be summarised as follows:

1. Collect data about projects' history.
2. Choose, develop, and compute metrics to base estimations on.
3. Use machine learning algorithms to build code churn estimation models.
4. Validate the performance of the models built in previous step.
5. Use descriptive analysis to study the models validated in previous step.

This approach results in development of code churn estimation models and identification of rules for managing code churn. The allocation of research questions to the approaches and solutions described above is shown on Figure 2.

The following subsections give an overview of the machine learning algorithms used in the dissertation and give a brief introduction to descriptive analysis.

1.3.1. Machine Learning

Machine learning is the study and application of algorithms that allow computers to learn from experience. Some of the more popular algorithm classes used by machine learning are regression algorithms, Neural Networks, Decision Trees, and Support Vector Machines [45,46,47]. The main classes of tasks solved by these algorithms are *estimation* (approximation of numeric values) and *classification* (systematic placement of items into categories). In this dissertation, both classification and estimation tasks are addressed. On one hand we consider the task of classifying XSLT files into those that will have low churn, medium churn and high churn in their next revision [22]. On the other hand, we are interested in estimating the code churn of an entire project during a period of time [44].

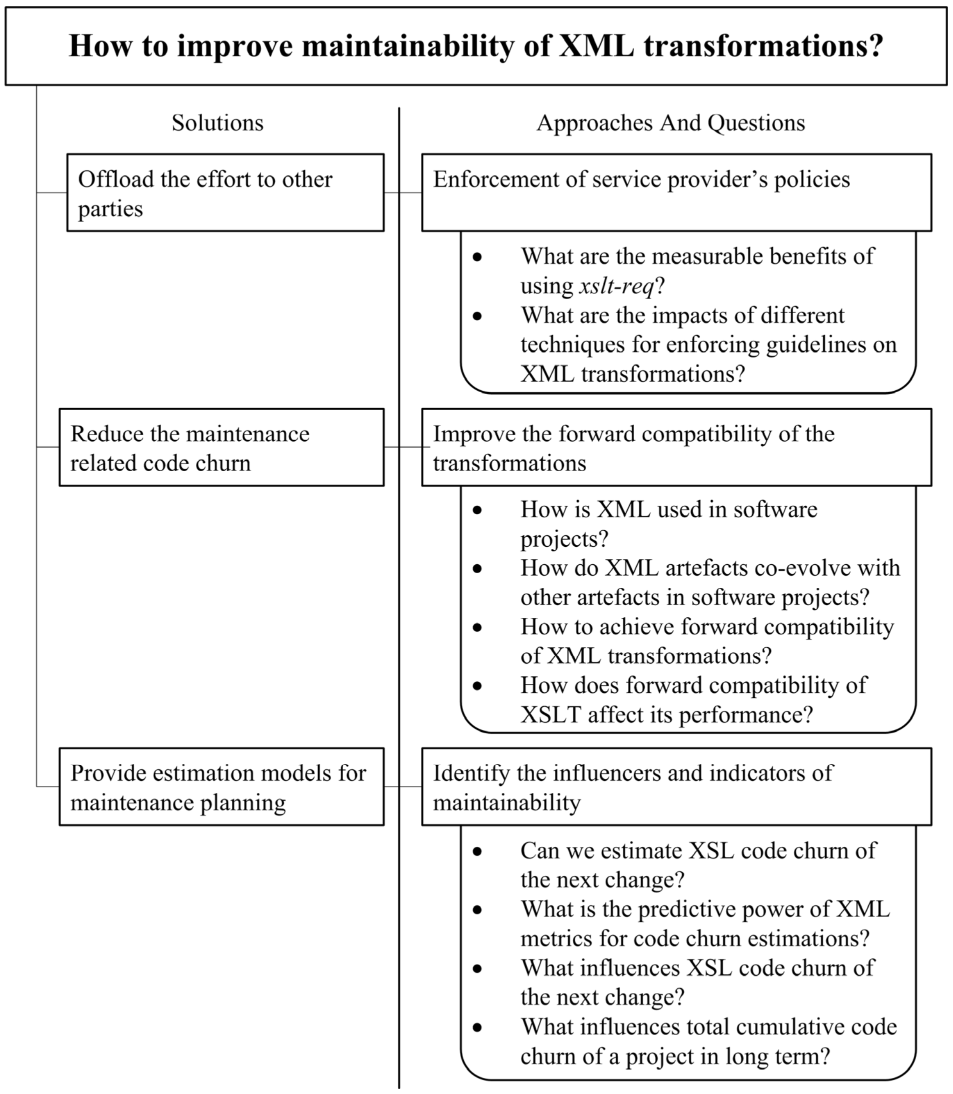

How to improve maintainability of XML transformations?
Solutions
Approaches And Questions
Offload the effort to other parties
Enforcement of service provider's policies
• What are the measurable benefits of using *xslt-req*?
• What are the impacts of different techniques for enforcing guidelines on XML transformations?
Reduce the maintenance related code churn
Improve the forward compatibility of the transformations
• How is XML used in software projects?
• How do XML artefacts co-evolve with other artefacts in software projects?
• How to achieve forward compatibility of XML transformations?
• How does forward compatibility of XSLT affect its performance?
Provide estimation models for maintenance planning
Identify the influencers and indicators of maintainability
• Can we estimate XSL code churn of the next change?
• What is the predictive power of XML metrics for code churn estimations?
• What influences XSL code churn of the next change?
• What influences total cumulative code churn of a project in long term?

Figure 2. Outline of the solutions and approaches discussed in the dissertation.

For classification we use Expectation Maximisation [48] (clustering) and Decision Trees (with constant values on the leaves). For estimation we use Single Layer Neural Networks and Decision Trees (with linear formulas on the nodes – also known as regression trees). In one article we also make use of Linear Regression and Logistic Regression algorithms, which are special cases of Decision Trees and Neural Networks algorithms. An introductory overview of these algorithms is given in the next subsections.

1.3.1.1.Expectation Maximisation

Expectation Maximisation is a data clustering technique similar to k-means and fuzzy c-means algorithms. The algorithm tries to minimise the differences between data points in a cluster by maximising the likelihood estimate. The algorithm results in a set of functions to indicate whether a data point belongs to a certain cluster or not (the data point is considered to belong to the cluster for which the function value is the highest). Details of Expectation Maximisation algorithm are described in [48].

1.3.1.2. Decision Trees

Decision Trees algorithms construct trees of formulas for the output features where each non-leaf node is a decision point based on the value of an input feature. An example of a simple three node Decision Tree is shown on Figure 3. In this tree the root node is a decision point based on "Number of Files" feature and the leaf nodes have linear formulas for calculating output feature named "Modified LOC".

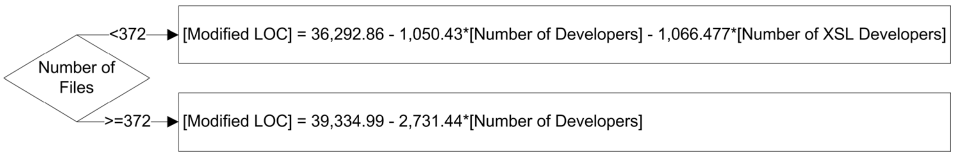

Number of Files
<372
>=372
[Modified LOC] = 36,292.86 - 1,050.43*[Number of Developers] - 1,066.477*[Number of XSL Developers]
[Modified LOC] = 39,334.99 - 2,731.44*[Number of Developers]

Figure 3. An example of a three node linear regression decision tree.

Decision Trees with linear regression nodes are highly suitable for identifying linear or multi-linear relations. The algorithm has limited ability for learning non-linear relations; however, it can approximate these with decision point nodes (splits in the tree). Building a Decision Tree with only one node and linear formula on that node is equivalent to building a model using Linear Regression algorithm.

1.3.1.3. Neural Networks

Neural Networks is a large class of biologically inspired learning algorithms. In our studies, we used three layer perceptron networks (Back-Propagated Delta Rule network) [49] as implemented by Microsoft SQL Server.

The Neural Networks algorithm used in our studies is, in principal, combining different logistic regression models, which offers improved performance for more complex relations than Decision Trees algorithm. On the other side, it is less suitable for identifying linear or multi-linear relations.

1.3.2. Descriptive analysis

Data mining algorithms produce models for solving given tasks; however, they do not provide insights into the mechanics discovered. Thus, descriptive analysis needs to be conducted on the models built by machine learning algorithms in order to unveil the mechanics involved.

For example, the following "black-box" descriptive analysis method can be used to identify features which influence the output value(s) of the models (also known as "influencers"):

1. Fixate the model's input features.
2. Calculate model's output O_0 for the selected set of features.
3. Choose one input feature F_i and modify its value.
4. Calculate model's output O_i for the modified set of features.
5. If O_0 and O_i differ, then F_i influences the model's output (and is an influencer).
6. Repeat steps 2-4 with different F_i-s to test whether these are influencers.

These steps allow us to identify influencers with possibility of false negatives. In other words, these steps might not identify all influencers. This "black box" approach can be modified to indicate the strength of influence and is often the only means of studying the relations between the input and output features.

Another method to analyze the models produced by machine learning algorithms is by inspecting their internal structure. In the case of Neural Networks, the formulas derived from the models are rather complex and difficult to explain. On the other hand, the design of Decision Trees is simpler, making the derived models easier to understand, for example in order to identify influencers. This fact is put into practice by the Decision Trees library used in this study, namely SQL Server Analysis Services Decision Trees library, which ranks input features by their influence toward the estimated value, meaning the dependency of the estimated value on the input feature. This ranking of input features is done by analyzing the tree, taking into account the regression coefficients of each feature and the decision points that are based on each feature.

In this dissertation, analysis of the internal structure of the estimation models is preferred over "black box" descriptive analysis. The "black box" approach is used only in cases where the model is too complex (e.g. in case of models trained using the Neural Networks algorithm).

1.4. Publications and Contributions

This dissertation is structured in the form of five complementary studies, which have been reported in separate research papers. The contributions and their mapping to research papers are as follows:

- Publication 1: A Study of Language Usage Evolution in Open Source Software [17]

    - Exploratory study of application and (co-)evolution of programming language usage in open-source software projects.
- Publication 2: Enforcing Policies and Guidelines in Web Portals: A Case Study [40]
    - Validation of the language for specifying the requirements to XML transformations.
    - Study of impact of different methods of reducing maintainability effort of XSLT.
- Publication 3: Designing Maintainable XML Transformations [21]
    - Development of detailed guidelines that help reduce maintainability effort of XSLT.
    - Verification of these guidelines and testing their impact on transformations' performance.
    - Analysis of the compatibility of changes to XSLT by change types.
- Publication 4: Predicting the maintainability of XSL transformations [22]
    - Predictive models of next revision code churn based on XSLT metrics.
    - Further exploratory development of guidelines that help reduce maintainability effort of XSLT.
    - Identification of influencers of XSLT code churn in software projects.
- Publication 5: Predicting Code Churn from XML Metrics [44]
    - Predictive models of yearly code churn based on XML and organisational metrics.
    - Identification of influencers of code churn in software projects.

1.5. Organisation of the Thesis

The dissertation is organized according to the five contributing studies as outlined above.

Chapter 2 summarizes the study of 22 open source software projects aiming at unveiling the role of XML in software development.

Next, Chapter 3 summarises a case study designed to test *xslt-req* and some general guidelines based on the components of forward compatibility. The case study is extended for evaluation and development of more detailed guidelines for XSLT by studying different scenarios of transforming XML. The summary of that study is presented in Chapter 4.

Having established that there is no performance penalty when following guidelines and having proposed a set of potential indicators of guidelines conformance and, thus, the coding effort; larger sets of projects were studied to unveil the relations between XSLT metrics and XSLT code churn for the next commit. The study, summarised in Chapter 5, confirmed the existence of such relation and helped to specify even more detailed guidelines for the design of maintainable XSL transformations. The last publication is summarised in Chapter 6 and takes a broader view by comparing yearly churn estimation models built on three different types of metrics: XML/XSL metrics, procedural/object-oriented programming language metrics, and organisational metrics.

The summary of each publication contains the aim of the study, the approach used in the study, the results of the study and related and future work related to the publication. The dissertation is concluded in Chapter 7, followed by bibliography (Chapter 8), acknowledgements (Chapter 9), and summary of the work in Estonian (Chapter 10). The original publications are in the Annex.

2. A STUDY OF LANGUAGE USAGE EVOLUTION IN OPEN SOURCE SOFTWARE

The first publication [17] studies the evolution of programming language usage in open source software (OSS) projects. By studying the code repository histories of 22 OSS projects over twelve years, the study answers the following questions:

- Which languages are used in OSS projects?
- How often different file types co-evolve?
- Which language combinations are used by the developers?
- How many artefact types do developers work with?
- Which artefact types are used by the new developers?
- Which artefact types are used by the more experienced developers?
- How have the answers to these questions changed during a decade?

Answers to these questions help software engineers understand the current state and trends in open source software development as well as the habits of open source software developers. This new understanding provides answers to research questions RQ2.1 and RQ2.2 presented in this dissertation.

2.1. Results

The study shows that the amount of code written in different languages differs substantially. Some of the other findings were:

- The most popular (i.e. widely used) language in OSS software projects is XML followed by Java and C.
- XML has increased its popularity steadily over the last decade while C has lost its high share to various other languages of which Java has been among the more popular ones. Despite becoming popular in just a few years, Java has not been able to grow its share significantly after 2007.
- Most Java developers work with XML, while only every second C/C++ developer works with XML.
- A significant increase of usage of XML and XSL was observed during recent years.
- The most commonly co-evolving files are usually of the same type.
- Most common co-evolving file types are: Java and XML; C and plaintext files; and C and Makefiles.
- Java and XML files (especially those of project specific types) are more likely than Java files and project definition files to be edited by the same person.
- JavaScript and CSS files most often co-evolve with XSL.

- A developer works with more than 5 different artefact types (or 4 different languages) in a project on average.
- New developers used fewer file types their first commits, even though most developers began with experience with multiple file types.

From the characteristics of developer language usage, we saw that not just knowing multiple languages is required from the developers, but developers must also understand different coding paradigms (e.g. procedural and object-oriented languages are often used side-by-side with rule and template based extensible languages). While in the 1990s they needed to know how to code in C and write Makefiles, the increased variety of languages used in newer projects and lack of distinct leaders in languages introduced the need to be familiar with multi-language development. These trends can be observed in Figure 4, which shows, how many developers use different languages across different years.

The high popularity of XML shown by the study highlights the need to understand and manage the evolution of XML in software projects. The fact that XML is mostly used in a supportive role suggests the possibility that high volume of non-XML (e.g. Java) code is written for manipulating XML. This possibility is further confirmed by the increasing popularity of XSL. Thus, the findings in this study support the need for studying means of reducing the maintainability of XML transformations (research questions RQ1.2 and RQ2.3 to RQ3.3). The high rate of co-evolution between XML artefacts and non-XML artefacts is a sign that evolution of XML and coding effort in a project are strongly related. This supports the motivation of building code churn estimation models based on features of XML code (research questions RQ3.1 to RQ3.4).

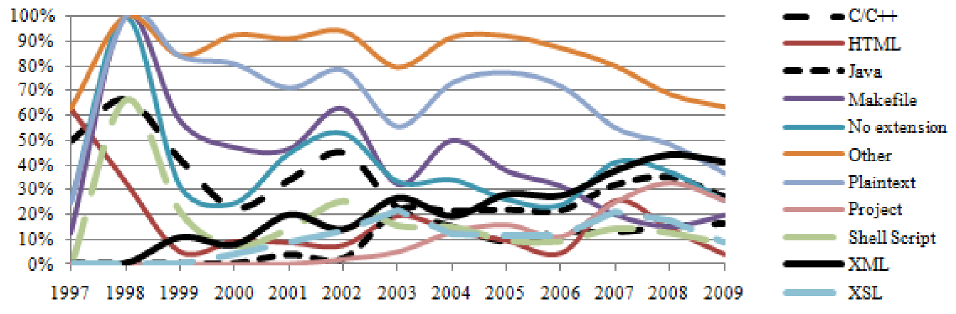


Figure 4. Proportion of developers generating different types of artefacts during different years.

7

2.2. Related Work

The idea of studying cross-file co-changes has been addressed by some research so far. However, these studies have often been language-specific and rarely look at different file types. Even studies encompassing multiple file types have been limited to specific file types. For example, Zimmerman et al. studied how lines of different files evolve in a project [50]. Their study is limited to textual files and focused on visualisation and clustering of blocks of lines based on their change history.

Weißgerber et al. have built a plug-in for Eclipse to show how likely different files are to be changed together [51]. Their tool does not exclude any files. However, they aim to visualise patterns emerging in specific projects regarding the co-evolution of files. They do not try to describe co-evolution on file type or artefact type level. As such their tool is useful for monitoring software development processes. In contrast, our study looked for more general patterns spanning through OSS software projects.

Dattero et al. conducted a survey during 2000–2001 and looked into differences by the developer gender [52]. They discovered that female developers are more likely to work with deprecated technologies. They also found that female developers tend to be less experienced and are familiar with only 2.53 languages as opposed to 3.25 languages male developers were familiar with. These numbers are similar to our findings, however, we also saw that the average number of different file types (usually representing different technologies) used by developers has decreased since the 1990s.

The different patterns of evolution of OSS have been outlined by Nakakoji et al. [53], who determined that there are three main types of OSS: exploration-oriented, utility-oriented, and service-oriented. These types determine how the software evolves and how the developers behave. In particular, their study shows that when projects move from one type of OSS to another, their development speed – which is related to the churn rate – is affected. The 22 OSS projects included in our study cover all three types of OSS identified by Nakakoji et al.

Open-source software repositories have been used for studying various aspects of software development like developer role identification (core or associate) [54], framework hotspot detection [55] and other. These works are complementary and help developing a better understanding of software development process and open-source software. It has also been shown that the number and size of open-source projects are growing at an exponential rate and open-source projects are becoming more diverse by expanding into new domains [56].

2.3. Future Work

The study reported above was made on the basis of 22 OSS projects. The representativeness of this set can be improved. For example, it was found that the amount of HTML code in these projects is smaller than the overall average reported by ohloh.net[2] – the largest analysed listing of open source projects. Also, the data extraction was made in 2009 and hence does not reflect more recent trends. Accordingly, a direction for future work is to perform a more comprehensive and up-to-date study. The characteristics of the ideal dataset would mirror those observed in ohloh.net. Since it is not feasible to incorporate all projects listed by ohloh.net (nor a large percentage of them), a sufficiently representative dataset should be extracted by means of appropriate statistical sampling techniques.

[2] http://www.ohloh.net

3. ENFORCING POLICIES AND GUIDELINES IN WEB PORTALS: A CASE STUDY

The second publication [40] evaluates a set of XML transformation and XML design guidelines on a web portal VabaVaraVeeb[3]. During the development of this web portal, the guidelines in question along with other policies set by the web portal's administrators were enforced both manually via developer documentation, API and code review, and semi-automatically via a policy[4] definition language called *xslt-req*. The study suggests that a combination of automatic verification and semantics extraction techniques can reduce the amount of manual checks required to enforce policies and guidelines for web presentation components.

The study was conducted as an analysis of a case study of efforts made to offload the effort of developing XML transformations to other parties in case of a public web service. In the case study, the quality of the transformations built by the independent parties was measured against the policies of the web service and a set of broadly defined guidelines for design of forward compatible XML and XML transformations. The two main focus points in the study were:

- The ratio of deviations from the service provider's policies detected automatically by the application of *xslt-req* language to all detected deviations from the service provider's policies.
- The changes in maintenance effort as the guidelines for forward compatible design were introduced to developers of transformation.

The study provides answers to dissertation research questions RQ1.1 and RQ1.2. The main contributions of the paper are:

- Validation of the *xslt-req* language for specifying policies on XML transformations.
- Study of impact of different methods of reducing maintainability effort of XSLT.

3.1. Results

The benefit of automatic verification techniques was found to be on the same level as the benefit of suggestive means (e.g. API changes and developer documentation with focus on guidelines). Therefore, suggestive means should not be undervalued. The automatic verification techniques were costlier than suggestive techniques to the portal's maintenance team as these techniques usually

[3] http://vabavara.eu

[4] Here, the term policy refers to a rule that must be followed and for which violation can be objectively defined, while the term guideline refers to a rule that should generally be followed, but for violations can not always be objectively asserted.

required human proofing or dealing with unforeseen issues. Even if the output of automatic verification were presented directly to the authors of the XML transformations, he or she might not be able to understand and solve the problems reported without the help from portals developers.

Less than half of deviations detected by automatic verification techniques were false positives (verification against *xslt-req* does not allow false negatives by virtue of its design). Automatic verification techniques detected less than 30% of all identified types deviations from policies. However, this 30% of types of deviations contained the most common deviations observed.

3.2. Related Work

There is extensive literature dealing with the enforcement of access control policies on XML content [57]. Policy definition languages proposed in this area allow one to attach access control policies to an XML document node and its descendants. In this sense, these policy definition languages share commonalities with *xslt-req*. However, they differ in several respects: First, access control policy languages focus on capturing the conditions under which a given XML node can be read or updated by a user. In contrast, they do not allow one to capture obligations such as "a given element must be displayed" or "an element must be displayed only in certain formats", both of which are key features of *xslt-req*. Nevertheless, *xslt-req* may benefit from ideas in [57] and in similar work, to improve its ability to capture access control requirements.

There is also significant literature related to enforcing accessibility and usability guidelines on web sites. For example, Vanderdonckt and Beirekdar [58] propose the Guideline Definition Language (GDL) which supports the definition of rules composed of two parts. The structural part designates the HTML elements and attributes relevant to a guideline. This part is expressed in a language corresponding to a limited subset of XPath. The second part (the evaluation logic) is a Boolean expression over the content extracted by the structural part. In contrast, *xslt-req* operates over XML documents representing the internal data managed by the portal, so that policies and guidelines are checked before the HTML code is generated.

The work presented in this paper is also related to the integration of services (possibly from multiple providers) at the presentation layer. This integration is supported by various portal frameworks based on standard specifications such as Java Portlets or WSRP [59]. More recently, Yu et al. [60] have proposed an event-based model for presentation components and a presentation integration "middleware" that enables the integration of services from multiple providers at the presentation layer without relying on specific platforms or APIs. However, these frameworks do not consider the enforcement of policies and guidelines as addressed in this paper.

3.3. Limitations of the Study and Future Work

The techniques presented in the paper have room for improvement along several directions. For example *xslt-req* could be extended to support the specification of rules based on patterns or XPath expressions. This way, *xslt-req* could be applied to more than just the static core structure. This would make it possible to allow all business layer services to have mandatory content or hidden content. In addition to extending *xslt-req*, the verification methods that use *xslt-req* need to be reviewed, as they currently assume that the document base structure is rigid.

The results showed that templates which automatically extract semantics from XML element names and values have been successfully used to achieve forward-compatible stylesheets and to enhance reuse, despite the fact that these techniques are not fail-proof. Making these techniques more robust and studying their applicability in a wider setting is an avenue for future work.

A limitation of this study is that it is based on a single scenario, which combined both semi-automatically-enforced and suggestive guidelines and policies. The study summarized in the next Chapter addresses this limitation in the context of the guidelines.

4. DESIGNING MAINTAINABLE XML TRANSFORMATIONS

The study reported in the third publication [21] is a step towards a more thorough and empirically-tested understanding of what is the impact of the application of the guidelines. The experimental evaluation on three scenarios shows that the proposed guidelines have a positive effect on the conciseness and extendibility of the transformations, while having no visible effect on the performance of the transformations (i.e. execution time).

The guidelines developed for the design of maintainable XSL transformations were:

1) Make use of common controls.
2) Make use of generic transformations.
3) Make transformations individually addressable and subscribable.
4) Make use of metadata in source documents.

Guidelines 1 and 3 seek to increase modularity, while guidelines 2 and 4 seek to increase abstraction, both of which are recognized conditions of forward-compatible software [26]. For detailed descriptions and examples of these guidelines, please consult the original paper [21].

These guidelines were put to test on three scenarios: a web service, a desktop application, and a business document transformation. Two versions transformations were created for each scenario: one designed before the introduction of the guidelines and another designed after the introduction of the guidelines. Lines of code needed for the transformation and the running time of the transformations were studied.

Additionally, in the web service scenario, the extensibility of the transformation was studied as new modules were introduced to both versions (guideline conformant and non-conformant version) of the transformations.

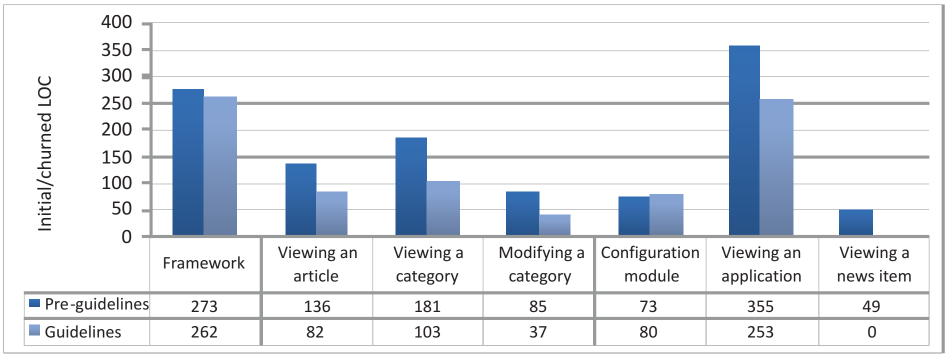
The study answers dissertation research questions RQ2.3 and RQ2.4. The main contributions of the paper are:

- Development of more detailed XSLT specific guidelines for achieving forward compatibility.
- Performance testing of guidelines conformant and non-conformant transformations.
- Experimental verification of guidelines' impact on code churn and XSLT metrics.

4.1. Results

The main results of the study are:

- The size of transformations was smaller when guidelines were applied to the transformations in all studied cases.



	Framework	Viewing an article	Viewing a category	Modifying a category	Configuration module	Viewing an application	Viewing a news item
■ Pre-guidelines	273	136	181	85	73	355	49
■ Guidelines	262	82	103	37	80	253	0

Figure 5. Impact of guidelines on transformation code churn.

- All cases had more complex expressions in guidelines conformant versions of transformations.
- Code churn for a feature addition was significantly lower when guidelines were applied in the case study. In fact, an addition of a new feature was accomplished with no presentation layer code churn (see Figure 5).
- The execution time for guidelines conformant and non-conformant transformations differed less than 3% without preference.
- The benefits of the application of the guidelines were most notable in the case of human-oriented output.

The paper also includes analysis of compatibility of different modifications of XSLT. The classification of the changes was based on a similar study on XML by Moro et al [61].

4.2. Related Work

Guerrini et al. studied the impact of XML Schema evolution on the validity of existing instances [62]. Their approach minimises the document fragments that need to be re-validated when an XML Schema changes. This makes it easier to identify forward and backward compatibility problems between documents that use different versions of a schema. This work and similar ones are orthogonal to our work since they do not deal with document transformations.

The first of our guidelines has similarities with output-driven approaches for XSL transformation design, such as the approach proposed by Lemmens and van Houben [63]. However, output-driven XSL transformation approaches do not have an explicit concept of common control.

There is a large body of research related to mapping-driven transformations, meaning transformations derived from a mapping between the elements in the source and the target schemas [64]. These mappings can be derived using automatic schema matching techniques [65] and may be visualized and edited by developers using graphical schema mapping tools. Graphical schema mapping tools are incorporated in enterprise application development platforms such as Microsoft BizTalk [16]. While these approaches enhance developer productivity, they are not designed to achieve change-resilience of the resulting transformations. In this respect, the guidelines studied in this paper are complementary to this body of work.

McDowell et al. proposed metrics that can be used to evaluate quality and complexity of XML Schema and conforming documents [66]. The principles they followed for selecting their metrics can be applied to selecting metrics for evaluation of other XML documents like XSL transformations. Additionally, the complexity of source and target documents of XSL transformations can be used to estimate the required complexity of the corresponding XSL transformation.

4.3. Future Work

The benefits derived from the guidelines are most notable in the case of transformations with human-oriented output (XHTML). One can therefore hypothesize that the benefits derived from the guidelines is greater for human-oriented output than for application-oriented output. A study of possible correlations between the purpose of the transformation and the usefulness of the guidelines is thus a desirable direction for future work.

Another desirable extension of this work is to consider other maintainability metrics than those based on lines of code or number of elements/templates. While these metrics provide an indication of maintainability, they do not provide a full picture. Other factors worth consideration include complexity, e.g. cyclomatic complexity [43].

Yet another aspect not considered in the study is that of enforcing the conformance of XSL transformations with respect to the proposed guidelines. Such an undertaking would require the definition of metrics for evaluating the degree of conformance of XSL transformations with respect to the guidelines, and techniques to pinpoint deviations with respect to the guidelines. This direction for future work could lead to quality metrics for XSL transformation with respect to maintainability.

5. PREDICTING THE MAINTAINABILITY OF XSL TRANSFORMATIONS

The study reported in the fourth publication [22] is a contribution towards building up an understanding of the factors affecting the maintainability of XML transformations – specifically XML transformations encoded using XSLT. The study was conducted on a dataset of XSLT file revisions from 15 open source software project repositories and answered the following questions:

- Can the code churn of the next revision of a given XSL transformation estimated from simple count metrics defined on XSLT?
- Are separate estimation models needed for business-oriented and presentation-oriented transformations?

The study used machine learning to build models for estimating next revision XSLT code churn. XSLT code churn was classified into three ranges: low churn (0–4 LOC), medium churn (5–16 LOC) and high churn (>16 LOC). Accordingly, classification models were built using Decision Trees, Clustering (Expectation Minimisation), Neural Networks, Logistic Regression (classification applied on testing set estimations), and Linear Regression (classification applied on testing set estimations) algorithms. The training and testing set were split randomly in a 70:30 ratio.

The study answers dissertation research questions RQ3.1, RQ3.2, and RQ3.3. The main contributions of the study are:

- Evidence that code churn of the next revision of a given XSL transformation can be estimated using models trained using machine learning algorithms on simple XSLT count metrics. These models can be built for both for business-oriented and for presentation-oriented transformations.
- Identification of a set of factors that indicate high future churn.
- Development of a set of guidelines for reducing XSLT next revision code churn.

5.1. Results

The main results of the study are:

- The maintenance effort measured in code churn can be predicted from simple XSLT metrics. The machine learning models built during the study were able to reliably identify transformations that are likely to undergo high churn.
- Estimation models based on both business type and presentation type transformations have higher or similar performance to models specific for either type.

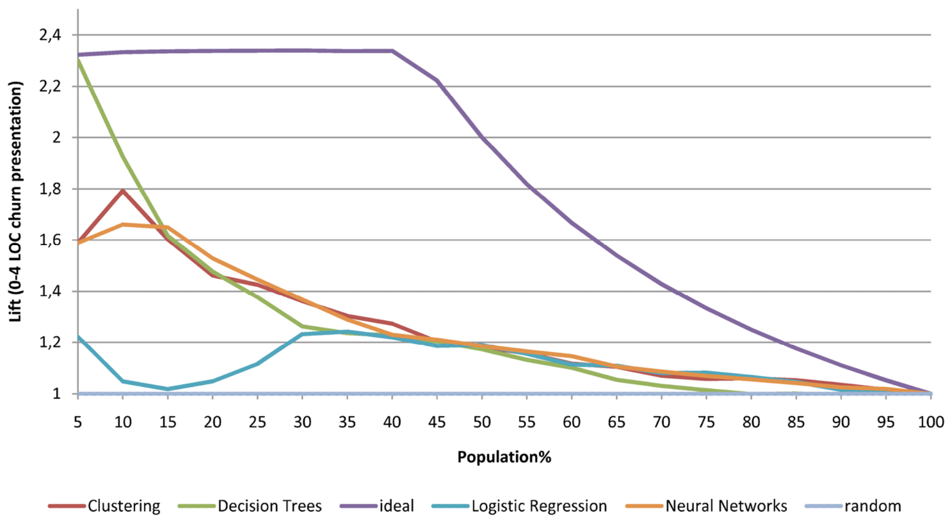


Figure 6. Lift at identifying ‘‘low churn’’ transformations in presentation type projects.

The estimation models proved to be especially good at identifying the most likely transformations belonging to different churn ranges. In particular, the top 5% of transformations ranked by the model trained using Decision Trees algorithm as “low churn” transformations had almost no false positives. This is shown by the fact that the models lift was close to the lift of a perfect model at 5% population (see Figure 6).

Descriptive analysis was applied on the models in order to unveil influencers of code churn. Based on the identified influencers, we formulated the following guidelines for reducing XSLT code churn:

1. Complex expressions (which were found not to influence code churn) should be used instead of simple expressions (which were associated with high churn).
2. Templates with ‘‘match’’ expressions (and if needed, ‘‘mode’’ attribute) should be preferred over <choose> and <when> constructs as high number of <when> elements and long <choose> constructs are associated with high churn.
3. Reoccurring constants should be stored as parameters or variables and used through these if possible. The study showed that a high number of text nodes characterise high churn while use of inline expressions and parameters encourages lower churn.
4. Parameter definitions should be kept short (less than 6 lines of code).
5. XSLT code should be commented. At least 9% of nodes being comments was identified a preferred ratio as it reduced high churn probability to a mere 5%.

The study also presented an online XSLT code churn estimation web service. The service estimates the next-revision code churn of a given XSLT transformation based on pre-trained models.

5.2. Related Work

Our work falls under the umbrella of a body of research aiming at mining software repositories in order to derive predictive models of software properties (e.g. predicting and locating defects, predicting changes, etc.). For example, it has been shown that high relative code churn is a predictor of system defect density [35]. Despite the rich body of work in this field [67], we are not aware of any technique that deals specifically with XSLT. In fact, almost none of the techniques developed in this field deals with XML and related languages.

The techniques we use are reminiscent of those used by Ratzinger, et al. [68] to mine software repositories in order to identify future refactoring of Java code. They predict the number of future refactorings of files in Java projects in short time-frames. However, they do not try to identify maintenance effort of a single refactoring, which is the case in our study.

There have been other attempts to identify high churn modules based on the code structure and program control flow. In one study by Khoshgoftaar et al. [41] code churn due to bug fixes is analyzed. In that study they used Discriminant Analysis to build accurate models for classification of fault prone modules. It is worth noting that the error rates they achieved are not directly comparable to our study as we used three classes instead of just two and we built models on a set of projects not a single project, which will have more consistent development practices.

Graphical schema mapping tools are incorporated in enterprise application development platforms such as Microsoft BizTalk [16]. While these approaches enhance developer productivity, they are not designed to achieve change-resilience of the resulting XSL transformations. In fact, the idea of these tools is that the XSL transformations are re-generated whenever a change is made to the mapping. This approach is not suitable for all applications as evidenced by the considerable number of manually developed XSL transformations found in commercial and open-source projects. In this respect, the guidelines unveiled in our study are complementary to this body of work.

There are some interesting research studies that have addressed the question of whether or not different project types evolve differently. For example, Vaucher and Sahraoui studied the evolution of software libraries [69]. They used neural networks and regression trees to build estimation models on software projects of two types: libraries and applications. They found that changes to well structured libraries are more localised than changes to applications. The machine learning algorithms employed in their study created relative code churn estimation models with higher predictive power on library-type projects.

5.3. Future Work

In this study only simple, mostly count-based metrics were used as input features. Thus, it is possible that the predictive power of the classification models could be improved by defining and including more complex metrics describing higher level features of XSLT. For example, in mainstream programming languages, metrics based on control-flow graphs, such as cyclomatic complexity, have shown to be highly correlated with maintainability [70]. However, it is not straightforward to adapt metrics based on control-flow graphs to XSLT. Other metrics worth considering are organizational and project metrics (e.g. size and experience of the development team, age of the project), which could turn out to be complementary to the code metrics studied in this paper.

Another avenue for future work is to design automated refactoring techniques in order to improve the maintainability of XSL transformations. To this end, we plan to identify common types of changes in XSL transformations and develop techniques to determine which change types can be applied to a given transformation in order to improve its maintainability.

6. PREDICTING CODE CHURN FROM XML METRICS

In the last publication [44] the effect of code and organisational metrics on long term code churn is studied. The study aims to answer following questions:

- What is the relative performance of models built to estimate the yearly cumulative code churn of a project based on: 1) XML/XSLT code metrics; 2) imperative/object-oriented programming language metrics; 3) organisational metrics; 4) organisational metrics and code metrics combined?
- What are the main influencers of yearly cumulative code churn in a software project?

The study involved applying Decision Trees and Neural Networks algorithms on 8 open source project repositories. The performance of the models was cross-validated using 7:1 split of the projects. The aim of the study was to compare the performance of churn estimation models trained on different sets of code metrics and organisational metrics. Only metrics extracted from revision control systems were used in the study.

The study addresses dissertation research question RQ3.4. The main contributions of the study are:

- Mapping of 9 common procedural and object-oriented language metrics to XML or XSL metrics.
- Confirmation that code and organisational metrics can provide for accurate estimations of yearly code churn in software projects.
- Evidence that XML metrics have higher predictive power than procedural and object-oriented language metrics when estimating project's yearly code churn.
- Development of highly accurate (mean relative error less than 1%) estimation model for yearly code churn estimations based on organisational metrics.
- Identification of strong influencers of code churn.

6.1. Results

The main results were:

- Organisational metrics provide for training models for code churn estimation at mean relative error less than 1% and mean normalised absolute error less than 0.01 and Pearson correlation close to 1.
- Combining organisational metrics with XML metrics improved estimation accuracy (absolute error less than 3400 lines of code) for confidence levels between 0.80 and 0.95. For higher confidence values, models based on only organisational metrics were superior in higher confidence ranges. Best models and error ranges at common confidence levels are shown in Table 1.

Table 1. Best models by confidence levels.

Confidence	Best Model	Error Range
0.70	Decision Trees all features	±300 LOC
0.80	Decision Trees organisational and XML	±600 LOC
0.90	Decision Trees organisational and XML	±1 400 LOC
0.95	Decision Trees organisational and XML	±3 400 LOC
0.99	Decision Trees organisational	±17 200 LOC
0.999	Decision Trees organisational	±83 000 LOC

- The strongest identified churn influencers are number of files of different types, project revision number, and active historical team size (total and by experience in the project with different languages).
- XML metrics were ranked as stronger influencers of code churn than object-oriented or procedural language metrics.

6.2. Related Work

Nagappan et al. used metrics from organisational structure to identify failure-prone binaries with great success [71]. They used more organisational information than available from version control systems. For example, information about not committing organisation members and organisational hierarchy was used. Their results also showed that code churn is the second best predictor of failure-prone binaries.

One of the earliest attempts to estimate code churn was made by Khoshgoftaar et al. [41]. In their study, they used multiple source code structure and control-flow metrics to train models for classifying software modules by future debug churn levels (high or low), which was used to determine whether a software component is fault-prone or not. In our research we estimate absolute churn values of projects instead of identifying software modules with high churn.

Zhou, et al. used linear regression to investigate relations between object-oriented design metrics and maintainability in open source Java projects [72]. In their paper, feature selection was done using linear regression. A similar approach can be used to identify interactions between features for feature construction purposes. Alternatively, the approach outlined by Smith et al. [73], which uses genetic algorithms to construct and select features, could also be employed for this purpose.

Our work is to some extent related to a large body of research in the area of software cost estimation [74]. The main differentiator is that software cost estimation aims at producing an estimate of a software project's cost (e.g. expressed in person-months) based on characteristics of the project – usually known ex ante – whereas the aim of our work is to make predictions of future coding effort based on a snapshot of a project, and specifically based on code metrics associated to file revisions in the project snapshot.

6.3. Future Work

We have shown that even though organisational models are highly accurate, using XML metrics to complement organisational metrics yields better results in some scenarios. We do also see a potential to build improved models based on combined feature sets given even larger training datasets. Building models on bigger datasets along with developing more sophisticated feature selection algorithms to reduce the amount of the required training data is an avenue for future work.

Another perspective for future work is to study the factors that affect the predictive power of source code metric and how these factors correlate with organisational metrics. Understanding this question could help us to build more sophisticated models, for example by employing more complex training algorithms with better feature selection methods.

7. CONCLUSION

The research discussed in this dissertation has resulted in (1) a better understanding of the usage of different languages in open source software projects, (2) development and validation of guidelines for the design of XSLT and XML transformations in general, and (3) short term and long term code churn estimation models. The studies reported in the original publications in the dissertation show that XML is the most often used language in software projects and is increasing in popularity. The most common use of XML is solution and project specific (e.g. in the form of domain specific language) – project definitions and configurations make up less than a third of XML files used in modern software development (RQ2.1). XML is not a stand-alone language; instead it co-evolves with other software development artefacts in about 20% of the cases (RQ2.2).

The research conducted in this doctoral project used a combination of a case study, controlled experiments and exploratory data analysis. The early case studies of application of some of the general guidelines for the design of forward compatible XML transformations and the XML policy language *xslt-req* showed almost doubled effectiveness of coding effort (RQ1.1 and RQ1.2). The following controlled experiment also showed that the performance of guidelines-conformant transformations is similar to the performance of transformations that do not conform to the guidelines (RQ2.4). Thus, there are no performance downsides to the use of the guidelines.

By studying the code churn of XSLT, a relationship between XSLT code churn and XSLT design was established (RQ3.1). Additionally, estimation models built by machine learning algorithms applied to data extracted from software repositories were found to achieve a mean relative error of less than 1% (RQ3.2). The introspection of yearly code churn estimation models showed that the strongest influencers of yearly code churn are organisational features (e.g. team size and experience), followed by XML and XSL related features. Code structure and object-oriented metrics were found to have the lowest influence on code churn.

The guidelines for designing maintainable XSLT were either identified by introspection of estimation models or developed by validating guidelines proposed based on the analysis of the properties of forward compatibility in controlled experiments. Some of the guidelines were the independent result of both of these methods. The resulting guidelines for designing maintainable XML and XSL transformations include the following (RQ2.3, RQ3.3, and RQ3.4):

- Reoccurring constants should be stored as parameters or variables and used through these if possible.
- Generic transformations should be preferred over specific transformations.
- Metadata in source documents should be taken advantage of.

- Complex (generic) expressions should be used instead of simple expressions when possible.
- Templates with ‘‘match’’ expressions (and if needed, ‘‘mode’’ attribute) should be preferred over <choose> and <when> constructs.
- Parameter definitions should be kept short (less than 6 lines of code).
- XSLT code should be commented.
- Transformations should be made individually addressable and subscribable.
- Long templates should be split into smaller ones to improve template reuse, addressability and subscribability.

The application of these guidelines results in slightly smaller files (~4% decrease in lines), lowered need and scope for changes needed to the files in the subsequent development steps (~20% lower code churn), and decreased yearly code churn in the project. The studies confirmed that these guidelines are applicable on different types of projects and transformation types.

In addition, the studies reported in this dissertation provided better understanding of XML and XSLT by linking different features of XML and XSLT code to features of object-oriented code and describing the effects of schema changes on transformation compatibility.

8. BIBLIOGRAPHY

[1] The World Wide Web Consortium (W3C). (2011, March) Extensible Markup Language (XML). [Online]. http://www.w3.org/XML/

[2] W3C HTML Working Group. (2002, August) XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). [Online]. http://www.w3.org/TR/xhtml1/

[3] RSS-DEV Working Group. (2008, June) RDF Site Summary (RSS) 1.0. [Online]. http://web.resource.org/rss/1.0/spec

[4] D. Winer. (2003, July) RSS 2.0 Specification (RSS 2.0 at Harvard Law). [Online]. http://cyber.law.harvard.edu/rss/rss.html

[5] M. Jazayeri, "Some Trends in Web Application Development, " in *Proceedings of the Workshop on the Future of Software Engineering (FOSE 2007)*, Minneapolis, MN, USA, 2007, pp. 199–213.

[6] Microsoft Corporation. (2010, April) Extensible Application Markup Language (XAML). [Online]. http://www.microsoft.com/downloads/en/details.aspx?FamilyID= 52a193d1-d14f-4335-aa86-c53193e1885d

[7] JBoss Community. Hibernate. http://www.hibernate.org/

[8] OASIS Web Services Business Process Execution Language (WSBPEL) TC. (2007, April) Web Services Business Process Execution Language Version 2.0. [Online]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[9] Health Level Seven International. (2010) HL7 V3 Messages Standard. [Online]. http://www.hl7.org/implement/standards/v3messages.cfm

[10] OASIS Open. (2006) ebXML. [Online]. http://www.ebxml.org/

[11] M. Bell, *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture.*: Wiley, 2008.

[12] World Wide Web Consortium. (1999, November) XSL Transformations. [Online]. http://www.w3.org/TR/xslt

[13] C. Kerer, E. Kirda, M. Jazayeri, and R. Kurmanowytsch, "Building and Managing XML/XSL-powered Web Sites: an Experience Report," in *Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC 2001)*, Chicago, IL, USA, 2001, pp. 547–554.

[14] C. Kerer, E. Kirda, and C. Krügel, "XGuide – A Practical Guide to XML-Based Web Engineering," in *Proceedings of Web Engineering and Peer-to-Peer Computing NETWORKING 2002 Workshops*, vol. 2376, Pisa, Italy, 2002, pp. 104–117.

[15] R. Wang, C. Yang, J. Xu, C. Yang, and X. Meng, "An XML-Based Interface Customization Model in Digital Museum, " *Transactions on Edutainment III*, vol. 5940, pp. 190–202, October 2009.

[16] D. Probert. (2008, February) BizBert. [Online]. http://www.bizbert.com/bizbert/2008/02/26/Understanding+The+BizTalk+Mapper+Part+12+Performance+And+Maintainability.aspx

[17] S. Karus and H. Gall, "A Study of Language Usage Evolution in Open Source Software," in *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR 2011)*, Honoulu, Hawaii, U.S.A., 2011, In print.

[18] W3C XSL Working Group. The Extensible Stylesheet Language Family (XSL). [Online]. http://www.w3.org/Style/XSL/

[19] W3C XSL Working Group. (2006, December) Extensible Stylesheet Language (XSL) Version 1.1. [Online]. http://www.w3.org/TR/xsl/

[20] W3C XSL Working Group and W3C XML Linking Working Group. (1999, November) XML Path Language (XPath) Verion 1.0. [Online]. http://www.w3.org/TR/xpath/

[21] S. Karus and M. Dumas, "Designing Maintainable XML Transformations," in *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR 2010)*, Madrid, Spain, 2010, pp. 137–145.

[22] S. Karus and M. Dumas, "Predicting the Maintainability of XSL Transformations," *Science of Computer Programming*, In press.

[23] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12–1990*, 1990.

[24] R. J. Martin and W. M. Osborne, *Guidance on Software Maintenance*. Washington, D.C, U.S.A.: U.S. Dept. of Commerce, National Bureau of Standards, 1983.

[25] E. B. Swanson, "IS "Maintainability": Should It Reduce the Maintenance Effort?," *SIGMIS Database*, vol. 30, no. 1, pp. 65–76, January 1999.

[26] C. Armbruster. (1999) Design For Evolution. [Online]. http://chrisarmbruster.com/documents/D4E/witepapr.htm

[27] J. F. Ramil and M. M. Lehman, "Metrics of Software Evolution as Effort Predictors – A Case Study," in *Proceedings of the International Conference on Software Maintenance (ICSM 2000)*, Washington, DC, USA, 2000, p. 163.

[28] J. J. Amor, G. Robles, and J. M. Gonzalez-Barahona, "Effort Estimation by Characterizing Developer Activity," in *Proceedings of the 2006 International Workshop on Economics Driven Software Engineering Research (EDSER 2006)*, Shanghai, China, 2006, pp. 3–6.

[29] B. W. Boehm et al., *Software Cost Estimation with Cocomo II*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[30] B. Boehm, C. Abts, and S. Chulani, "Software Development Cost Estimation Approaches – A Survey," *Annals of Software Engineering*, vol. 10, no. 1, pp. 177–205, November 2000.

[31] T. C. Jones, *Estimating Software Costs*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1998.

[32] P. C. Pendharkar and J. A. Rodger, "The Relationship Between Software Development Team Size and Software Development Cost," *Communications of the ACM*, vol. 52, no. 1, pp. 141–144, January 2009.

[33] G. A. Hall and J. C. Munson, "Software Evolution: Code Delta and Code Churn," *Journal of Systems and Software*, vol. 54, no. 2, pp. 111–118, October 2000.

[34] J. Munson and S. Elbaum, "Code Churn: A Measure for Estimating the Impact of Code Change," in *Proceedings. International Conference on Software Maintenance (ICSM 1998)*, Bethesda, Maryland, USA, 1998, pp. 24–31.

[35] N. Nagappan and T. Ball, "Use of Relative Code Churn Measures to Predict System Defect Density," in *Proceedings of the 27th International Conference on Software engineering (ICSE 2005)*, St. Louis, MO, USA, 2005, pp. 284–292.

[36] C. v. Koten and A. Gray, "An Application of Bayesian Network for Predicting Object-oriented Software Maintainability," *Information and Software Technology*, vol. 1, no. 48, pp. 59–67, January 2006.

[37] M. M. T. Thwin and T.-S. Quah, "Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics," *Journal of Systems and Software*, vol. 76, no. 2, pp. 147–156, May 2005.

[38] S. Karus, “Kasutajate poolt loodud XSL teisendustele esitavate nõuete spetsifitseerimine.,” Faculty of Mathematics and Computer Science, University of Tartu, Tartu, Bachelor’s Thesis 2005.
[39] W3C XML Schema Working Group. (2001, May) XML Schema. [Online]. http://www.w3.org/XML/Schema
[40] S. Karus and M. Dumas, “Enforcing Policies and Guidelines in Web Portals: A Case Study,” in *Proceedings of Web Information Systems Engineering – WISE 2007 Workshops*, Nancy, France, 2007, pp. 154–165.
[41] T. Khoshgoftaar, E. Allen, N. Goel, A. Nandi, and J. McMullan, “Detection of Software Modules with High Debug Code Churn in a Very Large Legacy System,” in *Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE 1996)*, White Plains, New York, 1996, pp. 364–371.
[42] T. Khoshgoftaar and R. Szabo, “Improving Code Churn Predictions During the System Test and Maintenance Phases,” in *Proceedings of the International Conference on Software Maintenance (ICSM 1994)*, Victoria, BC, Canada, 1994, pp. 58–67.
[43] H. Hayes, J. Liming, and Z. Liming, “Maintainability Prediction: A Regression Analysis of Measures of Evolving Systems,” in *Proceedings of the 21st IEEE International Conference on Software Maintanance (ICSM 2005)*, Budapest, Hungary, 2005, pp. 601–604.
[44] S. Karus and M. Dumas. (2010) Predicting Code Churn from XML Metrics. [Online]. http://arxiv.org/abs/1010.2354
[45] T. M. Mitchell, *Machine Learning*, 1th ed. New York, NY, USA: McGraw-Hill Book Co, 1997.
[46] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1th ed.: Springer, 2007.
[47] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2th ed., J. Gray, Ed. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2006.
[48] G. J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, 2th ed. New York, NY, USA: John Wiley & Sons, Inc., 2008.
[49] R. Hecht-Nielsen, “Theory of the Backpropagation Neural Network,” in *International Joint Conference on Neural Networks (IJCNN 1989)* , Washington, DC, USA, 1989, pp. 593–605.
[50] T. Zimmermann, S. Kim, A. Zeller, and J. E. J. Whitehead, “Mining Version Archives for Co-changed Lines,” in *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR 2006)*, Shanghai, China, 2006, pp. 72–75.
[51] P. Weißgerber, L. von Klenze, M. Burch, and S. Diehl, “Exploring Evolutionary Coupling in Eclipse,” in *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange (ETX 2005)*, San Diego, California, USA, 2005, pp. 31–34.
[52] R. Dattero and S. D. Galup, “Programming Languages and Gender,” *Communications of the ACM*, vol. 47, no. 1, pp. 99–102, January 2004.
[53] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, “Evolution Patterns of Open-Source Software Systems and Communities,” in *Proceedings of the International Workshop on Principles of Software Evolution (IWPSE 2002)*, Orlando, Florida, USA, 2002, pp. 76–85.
[54] L. Yu and S. Ramaswamy, “Mining CVS Repositories to Understand Open-Source Project Developer Roles,” in *Proceedings of the 4th International Work-*

shop on Mining Software Repositories (MSR 2007), Minneapolis, MN, USA, 2007, p. 8.

[55] S. Thummalapenta and T. Xie, “SpotWeb: Detecting Framework Hotspots via Mining Open Source Repositories on the Web,” in *Proceedings of the 2008 International Working Conference on Mining Software Repositories (MSR 2008)*, Leipzig, Germany, 2008, pp. 109–112.

[56] A. Deshpande and D. Riehle, “The Total Growth of Open Source,” in *Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software (OSS 2008)*, vol. 275, Milano, Italy, 2008, pp. 197–209.

[57] I. Fundulaki and M. Marx, “Specifying Access Control Policies for XML Documents with XPath,” in *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT 2004)*, Yorktown Heights, New York, USA, 2004, pp. 61–69.

[58] J. Vanderdonckt and A. Beirekdar, “Automated Web Evaluation by Guideline Review,” *Journal of Web Engineering*, vol. 4, no. 2, pp. 102–117, March 2005.

[59] C. Wege, “Portal Server Technology,” *IEEE Internet Computing*, vol. 6, no. 3, pp. 73–77, May/June 2002.

[60] J. Yu et al., “A Framework for Rapid Integration of Presentation Components,” in *Proceedings of the 16th International Conference on World Wide Web (WWW 2007)*, New York, NY, USA, 2007, pp. 923–932.

[61] M. M. Moro, S. Malaika, and L. Lim, “Preserving XML Queries During Schema Evolution,” in *Proceedings of the 16th International Conference on World (WWW 2007)*, Banff, Alberta, Canada, June 2007, pp. 1341–1342.

[62] G. Guerrini, M. Mesiti, and D. Rossi, “Impact of XML Schema Evolution on Valid Documents,” in *Proceedings of the 7th ACM International Workshop on Web Information and Data Management (WIDM 2005)*, Bremen, Germany, 2005, pp. 39–44.

[63] P. Lemmens and G.-J. Houben, “XML to XML Through XML,” in *Proceedings of the World Conference on the WWW and Internet (WebNet 2001)*, Orlando, Florida, 2001, pp. 772–777.

[64] H. Jiang, H. Ho, L. Popa, and W.-S. Han, “Mapping-Driven XML Transformation,” in *Proceedings of the 16th international Conference on World Wide Web (WWW 2007)*, Banff, Alberta, Canada, 2007, pp. 1063–1072.

[65] E. Rahm and P. A.Bernstein, “A Survey of Approaches to Automatic Schema Matching,” *The VLDB Journal*, vol. 10, no. 4, pp. 334–350, December 2001.

[66] A. McDowell, C. Schmidt, and K.-b. Yue, “Analysis and Metrics of XML Schema,” in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP 2004)*, Las Vegas, Nevada, USA, 2004, pp. 538–544.

[67] H. Kagdi, M. L. Collard, and J. I. Maletic, “A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution,” *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 19, no. 2, pp. 77–131, March 2007.

[68] J. Ratzinger, T. Sigmund, P. Vorburger, and H. Gall, “Mining Software Evolution to Predict Refactoring,” in *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, Washington, DC, USA, 2007, pp. 354–363.

[69] S. Vaucher and H. Sahraoui, “Do Software Libraries Evolve Differently Than Applications?: An Empirical Investigation,” in *Proceedings of the 2007 Symposium on Library-Centric Software Design (LCSD 2007)*, Montreal, Canada, 2007, pp. 88–96.

[70] W. Li and S. M. Henry, “Object-Oriented Metrics Which Predict Maintainability,” *The Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, November 1993.

[71] N. Nagappan, B. Murphy, and V. Basili, “The Influence of Organizational Structure on Software Quality: An Empirical Case Study,” in *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*, Leipzig, Germany, 2008, pp. 521–530.

[72] Y. Zhou and B. Xu, “Predicting the Maintainability of Open Source Software Using Design Metrics,” *Wuhan University Journal of Natural Sciences*, vol. 13, no. 1, pp. 14–20, February 2008.

[73] M. G. Smith and L. Bull, “Feature Construction and Selection Using Genetic Programming and a Genetic Algorithm,” in *Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2003)*, Essex, UK, 2003, pp. 229–237.

[74] M. Jorgensen and M. Shepperd, “A Systematic Review of Software Development Cost Estimation Studies,” *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 33 –53, January 2007.

9. ACKNOWLEDGEMENTS

I would like to thank everyone who has directly or indirectly helped me during my PhD studies. I would like to thank the staff and co-students at the Institute of Computer Science who have supported me in this quest in various ways. My special thanks go to my supervisors: professor Marlon Dumas, who agreed to supervise me even before he joined the institute, and docent Helle Hein.

I would also like to acknowledge that the work presented in this dissertation is partially funded by Swedbank and ERDF through the Estonian Centre of Excellence in Computer Science. The research was partly conducted during the visits of the first author to the software evolution and architecture lab at the University of Zurich. These visits were funded by the ESF DoRa 6 Program and Estonian Doctoral School in ICT. We thank the members of the lab for their valuable advice.

10. KOKKUVÕTE (SUMMARY IN ESTONIAN)

XML teisenduste hallatavus

Extensible Markup Language (XML) on laia levikuga andmete esitamiseks kasutatav keel. XMLi kasutatakse nii lõppkasutajatele andmete esitamiseks (nt. XHTML) kui ka süsteemisisesel suhtlusel ja andmevahetusel (nt. EDI, HL7). Selline laialdane kasutus erinevatel süsteemi tasemetel on kaasa toonud vajaduse teisendada XML dokumente ühelt kujult teisele. Selliste teisenduste haldamine võib aga olla väga kulukas. Käesoleva doktoritöö eesmärk on leida viise XML teisenduste haldamiskulude vähendamiseks, uurida XML-i ja XML teisenduste rolli tarkvara arenduses (sh. mõju loodava koodi mahule) ja tuvastada kodeerimismahtu määravaid indikaatoreid.

XML teisenduste koodi haldamiskulude vähendamisel on uuritud kahte suunda: 1) koodi haldamise ja arendamise delegeerimine partneritele ja 2) koodi hallatavuse suurendamine koodi edasiühilduvuse saavutamisega. Esimesel juhul on peaprobleemiks kõigi huvipoolte huvide kaitsmise tagamise võimalikkus ja rakendatavus. Selle probleemi lahendamiseks kontrolliti XML teisenduste poliitikate kirjeldamise keele *xslt-req* kasutatavust ja mõju poliitikate poolautomaatsel jõustamisel. Uurimuste tulemused näitasid, et *xslt-req* on kasutatav ning selle abil on võimalik automaatselt tuvastada ligi 30% poliitikate hälvetest. Samas on automaatselt tuvastatavad hälbed kõige sagedamini esinevad hälbed, mis kinnitab automaatsete kontrollide kasumlikkust.

Koodi edasiühilduvuse saavutamiseks kasutati kahte lähenemist: 1) edasiühilduvuse komponentide detailiseerimine ja kontrollimine ning 2) koodikulu (*code churn*) hindamise mudelite loomine ja analüüs koodikulu mõjutavate tegurite tuvastamiseks. Edasiühilduvuse komponentide detailiseerimine ja koodikulu hindamise mudelite analüüsi põhjal tuvastati järgmised peamised XSL arendamise juhised/mõjurid koodikulu vähendamiseks:

- Taasesinevad konstandid tuleb säilitada parameetrite (element <*param*>) või muutujatena (element <*variable*>), mitte kasutada kordusi koodis.
- Eelistada tuleb üldisi (erinevaid dokumendifragmente käitlevaid) teisendusi.
- Teisendused peavad kasutama lähtedokumendis olevaid metaandmeid.
- Keerulisi (üldiseid) XPath lauseid tuleb eelistada lihtsatele XPath lausetele.
- Elementi <*template*> atribuudiga „*match*” (ja atribuudiga „*mode*”) tuleb eelistada <*choose*> <*when*> konstruktsioonidele.
- Parameetrite definitsioonid peavad olema lühikesed (alla 6 koodirea).
- XSLT koodi tuleb kommenteerida.
- Teisendused tuleb luua eraldiseisvalt adresseeritavateks ja tellitavateks.
- Pikad teisendused tuleb jaotada väiksemateks.

Loodud juhiste järgimine ei mõjutanud teisenduste rakendamise jõudlust oluliselt kummaski suunas (mõju alla 3%), küll aga on juhiseid järgivad teisendused lühemad ja nende laiendamine vajab oluliselt väiksemat koodikulu. Uurimustöö käigus loodud aastase kumulatiivse koodikulu hindamise mudelid saavutasid suurepärase täpsuse – hinnangu keskmine suhteline viga jäi alla 1%.

Dissertatsioon koosneb sissejuhatavast osast, neljast avaldatud või avaldamisel oleva ning ühe retsenseerimisfaasis oleva artikli kokkuvõttest, järeldustest, kirjanduse loetelust, tänusõnadest ja eestikeelsest dissertatsiooni kokkuvõttest. Sissejuhatavas osas antakse ülevaade:

- peamistest dissertatsioonis kasutatud tehnoloogiatest ja standarditest (sektsioon 1.1 *Background*): XMLst, levinud XML teisenduste kirjeldamise keelest XSLT (*Extensible Stylesheet Language Transformations*), edasiühilduvusest (süsteemi võime käidelda oma tulevikuversioonide tarbeks loodud sisendit) ja kodeerimiseks kuluva vaeva mõõtmisviisidest. Viimastest on dissertatsioonis kasutatud koodikulu (*code churn*), mis on loodud, eemaldatud ja muudetud koodi mahu summa.
- XML teisendustega seotud problemaatikast ning tuuakse välja dissertatsiooni peamised teemad ja uurimisküsimused.
- uurimistöös kasutatud hallatavuse indikaatorite leidmise võtetest, masinõppe algoritmidest ja mudelite kirjeldava analüüsi võtetest.
- doktoritöö peamistest panustest.
- dissertatsiooni ülesehitusest.

Järgnevad artiklite kokkuvõtted:

1. Artikkel „*A Study of Language Usage Evolution in Open Source Software*" (avaldamisel) uurib keelte kasutust avatud lähtekoodiga tarkvaraprojektides. Uuritud 22 projekti enim kasutatud keel on XML, milles loodud failid arenevad enamasti koos teiste projekti tehistega (20% mitte-XML tehiste muudatused on kaasatud muudatusega XML failides). Uurimus lükkas ümber arvamuse, et XML on peamiselt kasutuses projektide kirjeldamiseks – Ant, Maven, Eclipse ja NetBeans failid moodustasid alla kolmandiku kõigist projektides esinevatest XML failidest.
2. Artikkel „*Enforcing Policies and Guidelines in Web Portals: A Case Study*" uurib XML teisenduste haldamise delegeerimisest tingitud probleeme ühe tarkvaraprojekti raames. See uurimus näitas, et keel *xslt-req* on kasutatav XML teisenduste poliitikate kirjeldamiseks ning sellel baseeruv automaatne poliitikate järgimise kontroll suutis tuvastada 30% erinevatest poliitika hälvetest. Täiendavalt leidis kinnitust edasiühilduvuse komponentide põhjal loodud üldsõnaliste XML ja XSL arendamise juhiste positiivne mõju XML teisenduste hallatavusele.
3. Artikkel „*Designing Maintainable XML Transformations*" jätkas artiklis „*Enforcing Policies and Guidelines in Web Portals: A Case Study*" uuritud XSL arendamise juhiste uurimist spetsialiseerudes detailsemate juhiste loomisele XSLT hallatavuse parandamiseks. Loodud juhiste jär-

gimine vähendas uuritud teisenduste koodi mahtu ning nende laiendamiseks vajaliku koodi mahtu ilma olulise mõjuta teisenduste jõudlusele.

4. Artikkel „*Predicting the maintainability of XSL transformations*" (avaldamisel) uuris XSLT koodi sisulise muudatuse tegemiseks vajaliku koodimahu hindamise võimalikkust XSLT koodi loendamismõõdikute põhjal. Loodud mudelid suudavad tuvastada koodimahukaid muudatusi vajavaid teisendusi ning on eriti tugevad teisenduste hulgast kõige koodimahukamaid muudatusi vajavate teisenduste leidmisel. Mudelite uuriv analüüs näitas mõningate selgepiiriliste seaduspärasuste esinemist koodi mõõdikute ja koodikulu vahel. Nende seaduspärasuste alusel loodi juhised hallatava XSLT koodi arendamiseks.
5. Artikkel „*Predicting Code Churn from XML Metrics*" (retsenseerimisel) uurib XML/XSL koodi, objektorienteeritud ja protseduurilise koodi ning organisatoorsete mõõdikute mõju aastasele kumulatiivsele koodikulule projektis. Loodud mudelid saavutasid suurepärase täpsuse – keskmine suhteline viga alla 1% ja 95% tõenäosusega on hinnangu viga väiksem kui 3400 koodirida. Olulisemad koodikulu mõjutavad tegurid on organisatoorsed ning arenduskeelest sõltumatud või seotud XML ja XSL-ga. Objektorienteeritud mõõdikud ning koodi ehitusel baseeruvad mõõdikud omavad vähest mõju aastasele kumulatiivsele koodikulule.

PUBLICATIONS

CURRICULUM VITAE

Name:	Siim Karus
Born:	04.04.1984, Tartu, Estonia
Citizenship:	Estonian
E-mail:	siim.karus@ut.ee

Education:

2007–2011 (estimated)	University of Tartu, doctoral studies in Computer Sciences
2010 spring	University of Zurich, exchange student
2005–2007	University of Tartu, Master of Technical Sciences (Computer Science), *cum laude*
2002–2005	University of Tartu, Bachelor of Technical Sciences (Computer Science), *cum laude*

Languages:
Estonian (mother tongue), English (excellent), Russian (basic), German (mid-level)

Main Research Topics:
XML, maintainability of source code

Organisational and Professional Activities:
- Microsoft Certified Trainer 2010–2011
- Microsoft Certified Professional
- Microsoft Certified IT Professional: Database Administrator 2008; Business Intelligence Developer 2008
- Microsoft Certified Technology Specialist: SQL Server 2008, Implementation and Maintenance; SQL Server 2008, Business Intelligence Development and Maintenance; Biztalk® Server 2006: Custom Applications

ELULOOKIRJELDUS

Ees- ja perekonnanimi:	Siim Karus
Sünniaeg ja koht:	04.04.1984, Tartu, Eesti
Kodakondsus:	Eesti
E-post:	siim.karus@ut.ee

Haridustee:

2007–2011 (eeldatav)	Tartu Ülikool, informaatika doktorantuur
2010 kevad	Zürichi Ülikool, külalisüliõpilane
2005–2007	Tartu Ülikool, magistrikraad tehnikateadustes (informaatika), *cum laude*
2002–2005	Tartu Ülikool, bakalaureusekraad tehnikateadustes (informaatika), *cum laude*

Keelteoskus:
Eesti keel (emakeel), inglise keel (kõrgtase), vene keel (algtase), saksa keel (kesktase)

Peamised uurimisvaldkonnad:
XML, tarkvara lähtekoodi hallatavus

Muu organisatsiooniline ja erialane tegevus:

- Microsoft Certified Trainer 2010–2011
- Microsoft Certified Professional
- Microsoft Certified IT Professional: Database Administrator 2008; Business Intelligence Developer 2008
- Microsoft Certified Technology Specialist: SQL Server 2008, Implementation and Maintenance; SQL Server 2008, Business Intelligence Development and Maintenance; Biztalk® Server 2006: Custom Applications