MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/03.20.19.58-TDI

CHARACTERIZATION OF CHANGES IN WEB SERVICES CONTRACTS BASED ON REPOSITORY MINING

Diego Benincasa Fernandes Cavalcanti de Almeida

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Prof. Dr. Eduardo Martins Guerra.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34P/3NHQ3T8>

INPE
São José dos Campos
2017

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6923/6921
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2017/03.20.19.58-TDI

CHARACTERIZATION OF CHANGES IN WEB SERVICES CONTRACTS BASED ON REPOSITORY MINING

Diego Benincasa Fernandes Cavalcanti de Almeida

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Prof. Dr. Eduardo Martins Guerra.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34P/3NHQ3T8>

INPE
São José dos Campos
2017

Cataloging in Publication Data

Almeida, Diego Benincasa Fernandes Cavalcanti de.

Cutter Characterization of changes in web services contracts based on repository mining / Diego Benincasa Fernandes Cavalcanti de Almeida. – São José dos Campos : INPE, 2017.

xxiv + 84 p. ; (sid.inpe.br/mtc-m21b/2017/03.20.19.58-TDI)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2017.

Orientador : Eduardo Martins Guerra.

1. Software repository mining. 2. Web service 3. Service contract. 4. Software adaptation. 5. Web service planning I. Título.

CDU 000.000

ATENÇÃO! A FOLHA DE APROVAÇÃO SERÁ INCLUIDA POSTERIORMENTE.

Mestrado em Computação Aplicada

"The way is open to everyone, and if some win and achieve what they want, it is not because they are predestined, but because they have forced the obstacles with boldness and tenacity."
(translated from Portuguese)

Henrique Coelho Neto
in *"Breviário Cívico (Energia)"*, 1921

Dedicatory

To Paula and Sofia, the family that God blessed me with.

ACKNOWLEDGEMENTS

To God, for blessing me, giving me wisdom and confidence in my own abilities.

To my lovely wife Paula, for being always by my side, encouraging me and giving me the necessary strength to keep going on.

To my future daughter Sofia, for being the biggest motivation for me to go anywhere I want.

To the Brazilian Army Geographic Service Bureau (DSG), for believing in my abilities and designating me to take this master degree course.

To my advisor, for always being supportive and helping me in every step towards the conclusion of the research.

And, finally, to everyone that stood by me with a sincere friendship.

Thank you all. You have made me a better professional and a better person.

God bless all of us.

ABSTRACT

During a software system life cycle, project modifications occur for different reasons, either for natural evolution or requirements readjustment. Regarding web services, communication contracts modifications are equally common, which induce the need for adaptation in every system node, from the service consumers to the providers. More significant those changes are, greater the efforts required for this adjustment. To help reducing the contracts changing impact over software source code, easy-to-adapt systems can be designed in order to minimize the application remodeling effort. However, to make this approach possible, it is necessary to understand how those contract changes occur, analyzing the most common modification types and how often they happen. In this sense, this dissertation undertakes an evaluation of the change history of different open-source projects whose web service contracts are defined using documents in Web Service Description Language (WSDL) format. Using software repository mining with MetricMiner tool, the behavior of four modification types (addition, removal, relocation and refactoring) that occur to four XML element types (`xs:element`, `xs:attribute`, `xs:complexType` and `xs:import`) of contracts schemas was analyzed, in a universe of 139 projects whose source-codes are hosted at GitHub. As a result of this study, conclusions were that modifications of types addition and removal were more frequent than the others and take place in about 20% of verified revisions, and that a great amount of commits – the act of recording file changings to the repository and creating a new file revision – are related to a small number of changings in contracts. Such results indicate that modifications tend to be spread in many revisions and that a significant amount of changes are related to inclusion or exclusion of exchanged information in contracts. Conclusions obtained serve as input to the planning of new web services and to the maintenance of existing ones, giving important knowledge about services evolution which helps reducing or even avoiding excessive adaptation effort of both clients and provides with the natural evolution of contracts.

Keywords: Software repository mining. Web services. Service contract. Software adaptation. Web service planning.

CARACTERIZAÇÃO DE MUDANÇAS EM CONTRATOS DE SERVIÇOS WEB BASEADA EM MINERAÇÃO DE REPOSITÓRIO

RESUMO

Durante o ciclo de vida de um sistema computacional, modificações no projeto ocorrem por diferentes motivos, quer sejam por necessidade de evolução ou para readequação aos requisitos. No que diz respeito a serviços web, modificações nos contratos de comunicação são igualmente comuns, o que causa a necessidade de adaptação de todos os agentes do sistema, desde os consumidores até os provedores dos serviços. Quanto mais significativas forem tais mudanças, maior será o esforço necessário para o ajuste. Para reduzir o impacto das alterações dos contratos sobre o código-fonte das aplicações, sistemas mais adaptáveis podem ser desenvolvidos de modo a minimizar o esforço de remodelagem da aplicação à nova versão do contrato. Contudo, para que tal abordagem seja possível, é necessário entender como tais mudanças em contratos ocorrem, analisando os tipos mais comuns de alterações e a frequência com que acontecem. Neste sentido, esta dissertação realiza uma avaliação do histórico de mudanças de diferentes projetos cujos contratos de serviços web são definidos por meio de documentos em formato Web Service Description Language (WSDL). Utilizando mineração de repositório com a ferramenta MetricMiner, foi analisado o comportamento de quatro tipos de modificações (adição, remoção, realocação e refatoração) que ocorrem em quatro tipos de elementos XML (`xs:element`, `xs:attribute`, `xs:complexType` e `xs:import`) dos esquemas dos contratos, num universo de 139 projetos cujos códigos-fonte estão hospedados no GitHub. Como resultado deste estudo, concluiu-se que modificações dos tipos adição e remoção são bem mais frequentes que as outras e que ocorrem em cerca de 20% das revisões verificadas, além de que grande parte dos *commits* – ato de gravar no repositório alterações em arquivos, criando novas revisões dos mesmos – estão relacionados a poucas alterações dos contratos. Os resultados indicam que as modificações tendem a se espalhar por várias revisões e que parcela significante de mudanças está relacionada com inclusão ou exclusão de informações trafegadas em contratos. As conclusões obtidas servem de insumo ao planejamento de novos serviços web e de manutenção dos já existentes, fornecendo conhecimento importante sobre a evolução dos serviços que auxilia a reduzir ou mesmo evitar esforço demasiado de adaptação tanto de clientes quando de provedores quando da natural evolução dos contratos.

Palavras-chave: Mineração de repositório de software. serviços web. contratos de serviços web. Adaptação de software. Planejamento de serviço web.

LIST OF FIGURES

LIST OF TABLES

Page

LIST OF LISTINGS

Page

LIST OF ABBREVIATIONS

CONCAR	–	Brazilian National Cartography Committee
DSG	–	Brazilian Army Geographic Service Bureau
GIS	–	Geographic Information System
GWS	–	Geospatial Web Services
INDE	–	Brazilian National Spatial Data Infrastructure
INPE	–	National Institute for Space Research
MSR	–	Mining Software Repositories
OGC	–	Open Geospatial Consortium
REST	–	Representational State Transfer
SOA	–	Service Oriented Architecture
SOAP	–	Simple Object Access Protocol
W3C	–	World Wide Web Consortium
WSDL	–	Web Services Description Language
XML	–	eXtended Markup Language
XSD	–	XML Schema Definition
XSLT	–	eXtensible Stylesheet Language for Transformation

CONTENTS

Page

1 INTRODUCTION

Integration is an important trend in a complex business network. Many companies develop and use their systems to supply different needs, like production control, data management or communication, based on the concept of integrating diverse components into a single solution. At the Brazilian Army, for example, a system to manage troops in campaign called "C2 em Combate" is developed over a modular architecture, where each module is responsible for a specific task – graphical user interface, geographic information management and processing, sensor data gathering, among others. A popular solution to achieve integration is the use of a Service Oriented Architecture (SOA), which acts as the basis for the computational systems creation, where each service provides a piece of functionality to be integrated in a given process. SOA is a reference architecture which aims to create functional modules called services, with low coupling and favouring code reuse (SAMPAIO, 2006).

In distributed computer systems, services that provide information over computer networks are known as web services. "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network" (W3C, 2004a). To make this possible, data must be exchanged using standardized messages, created using a structure and a vocabulary understandable by both communication nodes. These definitions are established in documents often referred to as "contracts", as they create the rules for the correct composition and interpretation of messages. In other words, the contract defines how nodes should interact in a SOA environment.

Web services messages are created using standard document formats, and a large number of those messages are based on the eXtended Markup Language (XML). The contracts that define XML messages vocabulary and structure are also created using a standard, known as XML Schema Definition (XSD). In fact, the term "contract" in this context is also referred to as "schema". According to W3C (2004b), XSD declares the vocabularies shared by machines and that represent the communication rules established by people, defining the structure, content and semantics of the exchanged documents. Complementary to that, Web Services Description Language (WSDL) standard provides means to define the operations supported by a web service (WEERAWARANA. et al., 2001), thus being mandatory that clients also know the existence of WSDL documents before making requests. It is relevant to mention that those standards are defined by the World Wide Web Consortium (W3C).

During software system development life cycle, changes occur in source code, even by

the introduction of new features, or due to modifications in existing ones. In a system build upon web services whose contracts are defined before implementation, changes that happen in XSD contracts impact services providers and consumers. A change in the contract delivers a change in the message format, and both service providers and consumers must adapt to this in order to maintain the communication. These contract changes can affect different systems, which might be developed by different teams and even by different institutions. Therefore, the extent and frequency of those modifications can influence the rate that system nodes need to adapt, since changes in contracts are expensive as they can affect several systems. If adaptation is not properly performed, integration can be compromised.

A recent study revealed that changes in web services contracts tend to often occur and they do impact on source code (FRANÇA et al., 2015). However, no research has been undertaken to address those changes and classify them according to frequency and location of occurrence. As so, some questions remain unanswered, like: (Q1) *"How usual element addition/removal is?"*; (Q2) *"Modifications changes document semantics or happen due to refactoring?"*; (Q3) *"Are modifications well distributed among documents and commits or concentrated in fewer ones?"*. Despite those – and many other – questions, studies lack on trying to answer them.

Understanding the pace of changes in contracts can lead to a redesign of development practices or to the adoption of architectures capable of adapting to them. If those modifications are mapped – or measured – accordingly, one can undertake a wider contract adjustment to reduce the frequency of changes, ultimately leading to more stable or more adaptable client-server integration.

1.1 Objective

The goal of the present work is:

To study the evolution of XML document formats used by web services contracts during development life cycle, aiming to capture the most frequent kinds of modifications in XSD schemas used by WSDL documents and to describe the changing behavior of projects contracts.

1.2 Research methodology

To achieve the main purpose, some issues need to be addressed. Those issues are treated as intermediate goals, each of them detailed in this section.

In order to analyze the changes in WSDL documents, some mechanisms need to be defined. As so, simple metrics are established, such as number of XML elements and attributes. Those metrics help to evaluate the pace of modifications and might lead to a changing pattern detection.

The defined metrics need to be integrated into a software system, in order to extract the desired information from XSD schemas. Using Java programming language, the mechanism to extract those metrics were materialized as Java classes. At first, in a preliminary study, only the results of metric extraction over XSD files inside a few "test projects" (selected only by the criteria of having XSD files inside their repository) are presented, as well as some simple inferences over them, to verify the applicability of the analysis techniques defined. After that, in-depth analysis were performed to gather specific information about the type of each modification. This second study included a wider range of web services projects containing WSDL documents.

To extract the desired information for the study, it is necessary to apply the defined metrics in several versions of WSDL documents present in real projects. This work focused on projects that define or make use of web services, using XSD schemas to specify messages elements and their types, inside or outside WSDL documents. Taking this constraint into account, several projects from GitHub – a web-hosted version control system – were selected. Based on the research goals, a minimum number of changes in WSDL files was defined as a threshold to establish significant inferences.

Once the projects were selected, the implemented metrics were integrated into a tool named MetricMiner (ANICHE et al., 2013) that extracts metrics from several versions of files hosted at a GitHub software repository, generating a historical database of them. Using MetricMiner, it was possible to evaluate the evolution of the selected metrics. Furthermore, the generated metrics database can be used for more complex analysis in future works, such as changing pattern detection or modification tendencies.

After the previous results, a new Java class was implemented to check the modifications of WSDL and XSD files over two consecutive versions of them, classifying the types of changes that happen. Examples of changes are: renaming of an element, new element or attribute, and element removal. After implementation, the class was coupled to MetricMiner, in order to obtain such information for each consecutive pair of WSDL file revisions.

Finally, with all the results gathered, a detailed analysis over them was proceeded. The goal was to evaluate the changing frequency for each modification type, looking for common changing characteristics and for possible similarities of those characteristics over different projects. Such characteristics include, for example, the average amount of changes and of modified documents per commit (the act of recording file changings to the repository and creating a new file revision). Ultimately, this study aims to answer the following research questions:

- **RQ1** – What is the occurrence rate of each XSD modification type?
- **RQ2** – With which pace XSD modifications change document semantics?
- **RQ3** – How is the distribution of modifications among commits?

1.3 Relevance for computer science

Regarding web services, contract changes might – and usually do – impact on data retrieval, notably when the exchanged messages alter their schemas. Considering that any software system naturally evolves and changes during its life cycle, the development of a web service should take this fact into account, but understanding how those changes occur and how often they happen is crucial in this task. Changing in XSD schemas can be difficult to handle, as they usually affect project source code (FRANÇA et al., 2015) and may also require a large set of changes both in service consumers and providers.

Different solutions to contract versioning is proposed in the literature, but none of them tries to understand the motivation behind the changes or their behavior – they assume that the changes occur and give means to deal with them. For example, Leitner et al. (2008) proposes the use of a *service proxy* that makes the correspondence between client requisition and some available service version, but this solution forces every service version to be up and running and proxy to be updated every time service is updated. An alternative to these particularities is proposed by Frank et al. (2008): the middleware to be used has a compilation of distinct *service interface proxies*, created every time the service changes its interface. In this scenario, each proxy correlates client requests to one specific service version (and subversions), reducing proxy update frequency.

If a contract changing tendency is known beforehand, developers can design adaptive systems prepared to deal with contract changes through time, favouring

system development optimization and strengthening system requisites compliance. Moreover, information regarding the frequency of XSD modifications that do not usually change semantics can play an important role on the system design, and knowing which kind of modification is more likely to occur in elements with certain characteristics can help developers to prevent those changes. Finally, understanding the behavior of contract changes can direct the project development planning towards a better changing schedule, avoiding a numerous amount of unnecessary revisions, reducing the efforts of service nodes to readjust their systems to new service versions and promoting a more stable interoperability.

1.4 Relevance for geographic information science

Geospatial data is constantly being produced by many organizations and individuals around the world using different technologies and for several purposes, from cartographic mapping to knowledge discovery. This generates a huge load of geoinformation, which is usually large in digital size. Use of data from many providers can be done by importing a copy of them to the local file system or by remotely querying and retrieving them as needed. The former procedure is recommended when it is desirable to work offline, but it consumes disk space and often brings unnecessary data along with it. Besides, local copy is not automatically synchronized. The latter procedure is usually taken by distributed Geographic Information Systems (GIS), where no duplicated data is created, which keeps them always up-to-date but requires a permanent network connection to retrieve data. Another benefit of querying data remotely before their retrieval is the ability to select only information of interest before loading them into local systems, reducing disk storage usage and network traffic.

In order to provide data online, the Open Geospatial Consortium (OGC) defined specifications (OGC, 2006) to create web services for querying and retrieving raster and vector geospatial data over web – known as Geospatial Web Services (GWS). Such specifications are taken as standards for the development of the so-called OGC Web Services (OWS). OGC does not implement any of these, but specifies the requirements for services implementation, such as needed operations service should provide to work correctly. Examples of geospatial systems based on OGC standards are the Brazilian National Spatial Data Infrastructure (INDE) (CINDE, 2010) and the Brazilian Army Geographic Database[1].

[1] Available at:http://www.geoportal.eb.mil.br/index.php/bdgex-1/bdgex-generalidades.

Despite the fact that OGC defines standards for every part of OWS design and implementation, data provided by such services might be – and usually are – structured not following global standards. This is a common scenario, since distinct geospatial data providers often uses non-standardized infrastructures, which are designed according to their own requirements. To avoid data compatibility issues, initiatives try to create standardized data structures, like those proposed by the Brazilian Army Geographic Service Bureau (DSG) and homologated by Brazilian National Cartography Committee (CONCAR) regarding cartographic mapping and inherent vector data (DSG, 2010). However, even with those specifications and considering every possible scenario, data retrieved by different OWS providers might not be equally structured, and so GIS solutions and data aggregators need to address this problem when merging data from different providers.

Geospatial data specifications are not unmutable and is continuously evolving to address new incomings. Following these modifications are the web services contracts changing, that need to adapt to new versions of the specifications. Understanding the behavior of contracts evolution helps to develop services clients ready to deal with the modifications that rise among different revisions of the specifications. This is of utmost importance for GIS applications and for integrated data infrastructures like the Brazilian National Spatial Data Infrastructure (INDE) that uses GWS to maintain interoperability between data providers nodes and the central aggregator. With the information of the most common characteristics and frequency of modifications in GWS contracts, developers can implement easy-to-adapt GIS solutions in order to keep the quality of knowledge discovery using spatial data from distributed systems. Also, knowing beforehand how contracts tend to evolve helps their maintainers to design a more suitable evolution agenda to prevent communication breaks. This could be done by extending the present research and applying similar methodology and tools.

1.5 Relevance for the National Institute for Space Research (INPE)

Being a technology center for space research, INPE holds an extensive list of projects, from informatics to astronomy and remote sensing. Specifically regarding the latter, many studies focus on computational technologies for geoprocessing, developing libraries and softwares for this purpose.

The main computer library maintained by INPE regarding geospatial data is TerraLib (CÂMARA et al., 2008), which provides many geoprocessing tools that can be used for Geographic Information Systems (GIS) development. As an example

of its usage, INPE developed TerraView[2], an open-source GIS environment with a wide range of geoprocessing tools. On top of TerraLib, INPE builds solutions for environment monitoring and geospatial data manipulation and dissemination. Examples of these solutions are: TerraOGC, an extension that enables TerraLib to access and retrieve spatial data from OWS providers; TerraMA2, a platform to develop environment risk monitoring systems; TerraHidro, a system for distributed hidrological modeling. TerraLib and the systems build on top of it are extensively used, and researches using them are vastly found in the literature, like the ones proceeded by Crepani and Medeiros (2005), SILVA et al. (2007), Lima et al. (2012), Bendini et al. (2014), Barbosa and Silva (2014), Rosim et al. (2014), Rosim et al. (2012), among others.

OWS standards do not experience evolution in a regular basis, which can be verified checking OGC website. However, not all demands fit in those standards, as there are web services not compliant to OGC specifications. In case of changings in OGC WSDL schemas, every project that relies on them – including TerraLib with TerraOGC – might be impacted, and understanding the changes more likely to occur can help to adapt the systems to reflect those modifications. Also, similar interpretation can be obtained from non-OGC GWS.

1.6 Originality

Search for similar studies over the literature leads to very few related researches. Some of the most recent studies are described here as reference, but none of them have the same goals as this dissertation.

França et al. (2015) presents a study over the impact of web service contracts changing on the software system source code. Through historical data mining and analysis on software repositories, it gathers statistics for changing frequency of XSD files, along with information about the most usual modifications and its concentration by developer. Those metrics are defined by the authors and aim to establish the influence of XSD changes on source code, relating XSD modifications quantity with source code modifications quantity. As a final result, the authors deduce that contracts modifications are frequent and find evidences of huge impact on source code. That study enforces the need to better understand the contract modifications in order to develop web services prepared to deal with these changes. It is relevant to mention that the comparison between XSD modifications and source

[2]Available at: `www.dpi.inpe.br/terraview`.

code changings were done manually and with few projects, and so it is not able to generalize answers to any research questions listed at Section 1.2.

Qiu et al. (2013), endeavors a study on the evolution of schemas in relational databases, associating the modifications with the database queries created. The research classifies the modifications and also gathers some statistics to measure the impact of schema changing on those queries. It is not clear whether the authors use some kind of software to help with the analysis or not, but some proceedings seems to be similar to the ones that will be undertaken in the present work, like historical analysis, classification of changes and statistical comparison of modifications. Like França et al. (2015), it aims to "measure" the impact of schema changes at code-level and is related to this dissertation by the inner-observation of modifications and discrimination of different change types and occurrence frequency in schemas, regarding that the present study analyzes XSD schemas used by WSDL documents.

Papazoglou (2008) introduces a change-oriented service life cycle methodology to address the problems that arise from contracts versioning. It provides ways to allow services to be readjusted as changes occur, as well as common tools to reduce drawbacks and improve development agility. Its goal is to deal with contracts changes and overcome the problems and keep interoperability. Like the other researches, it does not intend to understand the modifications, but to accept their existence and to solve miscompatibility.

Romano and Pinzger (2012) present a research regarding web services evolution. Through the implemented *WSDLDiff* tool, the study aims to identify common characteristics in contracts evolution, by analyzing the modification behavior of some types of XML elements. Despite being very similar to the proposal of this dissertation, it does not extend much the analysis, concentrating the efforts on few changing types. Also, it only analyzes four web service projects, which is not enough to generalize the results. It can be said that this is a starting point for the present research.

Despite the aforementioned researches, few others could be found dealing with the changes in contracts of web services and trying to understand where and how those changes occur. With that said, this can be considered an original research.

1.7 Document structure

To organize the study, this dissertation is organized in several chapters, each one with a particular goal. The following lines describes each chapter.

Chapter 2 presents the knowledge basis that guides the study. It describes and explains needed concepts to understand the undertaken analysis and its results.

Chapter 3 shows the design, the methodology and the results of a preliminary study directed to introduce the main research as described at Chapter 1. This study defines simple metrics to be applied over XSD documents and uses them to collect information about the frequency of modifications of three XSD container types. Being an introductory research, it only considers a short number of projects that uses or defines XSD documents. With the obtained results, some research questions are answered, which supports the need of further investigations on contract changes.

Chapter 4 describes the following steps towards the achievement of dissertation goals. It discriminates the methodology applied, the code implementation needed and the results expected.

Chapter 5 presents the results obtained from the application of the methodology. It describes the analysis over data gathered from code implementation by answering the research questions defined in Section 3.2.

Chapter 6 come up with the conclusions of the study based on the answers of the research questions given in Chapter 5. It also suggests future researches based on the results and methodology of this study

2 FUNDAMENTALS

This chapter presents the concepts that should be known in order to understand the analysis proceeded through the dissertation. The following sections give the necessary knowledge about topics underlying this research.

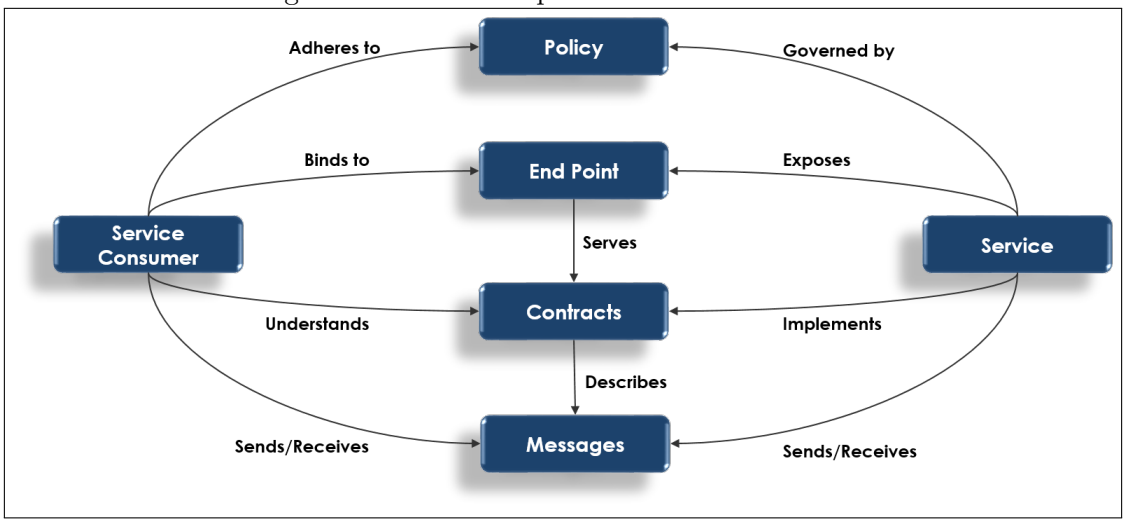
Section 2.1 shows the principles regarding the Service Oriented Architecture (SOA) and its use in computer technology. Section 2.2 and Section 2.3 describe two materializations of SOA principles. Section 2.4 explains the uses of XML in the context of SOA and web services. Section 2.5 presents a description of the different means that a web service can be developed. Finally, Section 2.6 gives an explanation about the techniques of mining software repositories.

2.1 Service Oriented Architecture (SOA)

In the context of architectures formed by several systems, efforts are directed to integration and collaboration. Reuse of already available data avoids reworking and redundancy of potentially incoherent information. In order to allow data sharing, systems were designed in such a way that a *consumer* could ask for data to a *provider*, assuming that the latter has the required information. The provenance of available data to eligible consumers creates a so-called *Service-Oriented Architecture* (SOA), where providers have well defined communication roles and interact with each other using standardized protocols, often called *contracts*. Figure 2.1 illustrates the main concepts of SOA.

Formally, SOA is defined as "any design or specification for sharing data and/or processes in a network or other computing environment" (CONNER; ROBINSON, 2007). As so, it presents an arrangement of enclosed, independent and well-established computational functionalities designed to provide some kind of knowledge to each other. To reach this, it follows a large set of best practices (PATIG, 2011), from which some could be highlighted: (a) *reusability* – service functionality must be generic enough to be used in scenarios different from those imagined for it at first; (b) *statelessness* – services undertake their entire processes with the information provided by its client only, without information being kept within it after a call-and-response procedure; (c) *standardized contract* – service and consumer must share a common "communication language", in order to provide a well-known technical access interface composed by operations and their structures; (d) *loose coupling* – contracts must be client-independent and implementation-independent; (e) *autonomy* – services have to be independent and not rely on the responses from

Figure 2.1 - SOA components and their relation



SOURCE: adapted from Rotem-Gal-Oz (2007).

other services; and (f) *discoverability* – services should be described by meta data, in order to be discovered and interpreted by clients.

In order to materialize SOA concepts and principles in computer systems, there are currently many technologies available. Most of them, however, are based on some kind of message exchange, where the message contents depends on the communication direction: if it is a request message, it must call for the required service functionality providing the necessary request parameters; if it is a response message, it should deliver the requested information in a way that the consumer can interpret and manipulate it by its own. Among the available technologies, *Simple Object Access Protocol* (SOAP) and *Representational State Transfer* (REST) are two of the most used ones regarding web services.

It should be pointed out that there are two main approaches for implementing web services: the "contract first" and the "contract last". According to Pautasso et al. (2008), "contract first" approach starts the service development by the specification of its interface, and "contract last" automatically generates contracts from previous implemented services. In "contract first" development, implementation follows a specification of what operations a service should provide, and changes in this specification might have huge impact on source code. That is the main reason why understanding modifications in such contracts is important to the service development and maintenance.

2.2 SOAP

According to W3C (2007), SOAP "is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment". It is a framework that enables systems to interoperate using customized and well-defined messages. Those, in SOAP, are structured as Extensible Markup Language (XML) documents, which presents one of its greater advantages as many different systems could create and understand messages easily.

SOAP for web services are described using Web Services Description Language (WSDL) documents that inform its name, lists its available functionalities and details needed request parameters and response values. WSDL is written in XML and is a W3C standard (WEERAWARANA. et al., 2001), but its use is not limited to SOAP, although only a few bindings are described in the official documentation. It can use distinct XSD documents to describe the elements used for service operations definition or include the schema in it as XSD. A brief example[1] of a WSDL file is presented in Listing 2.1.

Listing 2.1 - Example of WSDL document

```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>

   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
```

[1] Extracted from: https://www.tutorialspoint.com/wsdl/wsdl_example.htm.

```
29            <soap:body
30               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
31               namespace="urn:examples:helloservice"
32               use="encoded"/>
33         </input>
34
35         <output>
36            <soap:body
37               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
38               namespace="urn:examples:helloservice"
39               use="encoded"/>
40         </output>
41      </operation>
42   </binding>
43
44   <service name="Hello_Service">
45      <documentation>WSDL File for HelloService</documentation>
46      <port binding="tns:Hello_Binding" name="Hello_Port">
47         <soap:address
48            location="http://www.examples.com/SayHello/"/>
49      </port>
50   </service>
51</definitions>
```

SOAP is commonly related to web services, but its use is not limited to this. SOAP messages can be exchanged through many different protocols as they are XML documents structured according to a XML Schema Definition (XSD) specification. Any protocol that can transport these messages can be used, including, but not limited to, Hypertext Transfer Protocol (HTTP).

Although it is a well established standard, SOAP has many limitations of use. Being XML documents, serialization and deserialization can consume a considerable amount of time in systems with a large amount of requests. Other fact is that as messages have to adhere to a contract (represented by XSD documents), communication might be broken if this contract changes, compromising interoperability. Nevertheless, in complex systems where stronger data structures are required, SOAP is the best choice as messages share a common semantic and will hardly be misunderstood due to distinct contracts.

The scenario where different contract versions are used can be bypassed using an Enterprise Service Bus (ESB), a middleware tool that handles heterogeneity and takes care of the message conversion between compatible contracts to maintain communication (CHRISTEN, 2009). According to Menge (2007), ESB is a "message-based, distributed integration infrastructure that provides routing, invocation and mediation services" which eases the interactions between distributed applications

and services in such a way that avoids communication failures. It is not a trivial task to design an ESB, as it must be modelled and implemented according to the applications involved in the composed solution.

2.3 REST

REST architecture is another type of communication bridge that makes use of named resources over the web. Those names are represented by fully qualified Uniform Resource Identifiers (URI), Uniform Resource Locators (URL) and/or Uniform Resource Names (URN). It differs from SOAP fundamentally by being based on HTTP requests and fully using its capabilities, and response messages do not have to be compliant to a predefined structural contract. As so, many serializations exists, like XML and JavaScript Object Notation (JSON).

Services that are fully compliant to the REST principles listed by Pautasso et al. (2008) are usually called RESTful. Those principles are: (a) resource identification through URI; (b) uniform interface; (c) self-descriptive messages; and (d) stateful interactions through hyperlinks. The use of those principles in a web service development allows a decentralized infrastructure to be constructed using common HTTP standards that are lightweight, well-established around the world and easy to implement and maintain. Those are only basic principles, and the complete set of them are listed and explained in details by Fielding (2000).

Another feature of RESTful services that should be highlighted is their capability of describing resource information through a variety of formats (PAUTASSO et al., 2008). Messages that carry on this data are self-descriptive and do not have do adhere to a service contract like SOAP services. Interpretation and usage of data provided by RESTful services must be treated by the client, and so it requests information using the most suitable serialization for its use (for example JSON, XML, PDF or plain text).

2.4 XML and web services

As stated earlier, web services use messaging protocols to exchange data. SOAP services relies on XML messages to communicate, while RESTful services can use this serialization but are not restricted to it. In both scenarios XML can be present, either as requirement or option.

XML stands for *Extensible Markup Language*. As its name suggests it is a format for creating documents hierarchically structured in order to store and transport data

across communication nodes. Being self-descriptive and flexible – as it does not use predefined tags – it can be easily adapted to the system needs. XML is derived from the Standard Generalized Markup Language, or simply SGML (ISO, 1986), and is a World Wide Web Consortium (W3C) recommendation (BRAY et al., 1998).

Inside XML documents, information is enclosed in containers called *tags*, and data described in tags are denominated *entities*. Needed entities are included in the XML message according to the requirements of every communication node and their structure, as well as the description vocabulary, should be known beforehand or provided by the message itself. In the latter case, it is commonly used a XML Schema Definition (XSD) document, which acts as a contract between the message sender and each receiver. With XSD, XML messages could be validated and considered well-formed or not by XML processors, avoiding problems that can compromise interoperability between applications. It is relevant to know that XSD is also a W3C recommendation (W3C, 2004b).

Listing 2.2 presents an example of a XML document that describes a ship order, while Listing 2.3 shows the XML schema of this document[2]. It is clear that XML and XSD provides organization to information being transported. The example illustrates that entities have their own definitions (name, type and attributes) and structural placement inside XML, which favors the integration enforcement between communication nodes.

Listing 2.2 - Example of XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="shiporder.xsd">
    <orderperson>John Smith</orderperson>
    <shipto>
        <name>Ola Nordmann</name>
        <address>Langgt 23</address>
        <city>4000 Stavanger</city>
        <country>Norway</country>
    </shipto>
    <item>
        <title>Empire Burlesque</title>
        <note>Special Edition</note>
        <quantity>1</quantity>
        <price>10.90</price>
    </item>
    <item>
        <title>Hide your heart</title>
        <quantity>1</quantity>
        <price>9.90</price>
```

[2]Source: W3Schools. Available at: http://www.w3schools.com/xml/xml_whatis.asp

```
    </item>
</shiporder>
```

Listing 2.3 - XML schema for the given XML example

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="shiporder">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="orderperson" type="xs:string"/>
                <xs:element name="shipto">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="name" type="xs:string"/>
                            <xs:element name="address" type="xs:string"/>
                            <xs:element name="city" type="xs:string"/>
                            <xs:element name="country" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="item" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="title" type="xs:string"/>
                            <xs:element name="note" type="xs:string" minOccurs="0"/>
                            <xs:element name="quantity" type="xs:positiveInteger"/>
                            <xs:element name="price" type="xs:decimal"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="orderid" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

As said before, SOAP services uses XML to exchange messages. But besides that, it also uses XML to fully describe the service, detailing each available operation on them and how to request data. This is done using Web Service Description Language (WSDL) documents, also a W3C recommendation (WEERAWARANA. et al., 2001). Hence, two scenarios can be distinguished: (a) request and response messages contains entities structured and defined by XSD documents; and (b) what can be requested from the service and how the response will be given is described using WSDL. WSDL are a special case of XSD, as it also defines things that can appear inside XML messages and how it should appear.

RESTful services are not based on XML, but can serialize messages in XML according to the client request. Due to this, they can use XSD to describe XML

entities like SOAP services. Being based on the concept of resources, RESTful services do not have the need to have a service description document like WSDL (but it can be used due to the flexibility of WSDL), and so the use of XSD is not as common as in SOAP services.

2.5 Web Service Development

As mentioned in Section 2.1, a web service can be developed following two opposite approaches: defining the contract in first place or automatically generating it after the service implementation. The first approach focuses on the transported data and on the message itself, and is used in scenarios where it is mandatory to have a message exchanging standard and well defined operations regardless the actual implementation. ON the other hand, the second approach focuses on service implementation and code writing, and produces contracts just for interoperability purposes.

The two aforementioned approaches are often known as "contract first" and "contract last", as stated by Pautasso et al. (2008). "Contract first" implementation derives from the specification of the operations that service needs to provide, whereas "contract last" establishment just exposes the source code classes and methods as WSDL. Considering that clients must be constantly compliant to contracts in order to consume the service, it is desirable that those contracts remains as much unchanged over time as possible (of course, some changes might occur), but only "contract first" approach allows the implementation of classes that deals with distinct contract versions (POUTSMA et al., 2007). Furthermore, "contract last" approach does not enable web services to reuse existing definitions, and also can create a large number of contract versions since each code modification can impose changings in the contract, compromising interoperability.

In a *contract-first* approach, logics and semantics regarding the data that should be exchanged are structured according to XML standard, with its definitions and particularities. This forces service providers and clients to suit their implementations to these characteristics. On the other hand, in a *contract-last* approach, contracts are derived from service implementation and generated automatically. According to Poutsma et al. (2007), while *contract-first* method looses coupling with implementation, *contract-last* technique brings some disadvantages, like: (i) particularities of the code language used to implement the service might not be easy adjustable to XML components, which might generate contracts hard to be understandable; (ii) any change in service source code can propagate unnecessary

modifications to the contract, making clients have to change their implementations to receive and interpret the messages.

Since "contract last" approach depends on the web service implementation model, code-writing patterns and source code modification rate, it is easy to infer that consumers of such services usually have to constantly adapt their systems to each contract version. Besides, analysis over the automatically generated WSDL documents do not represent a real service design modification analysis, which can only be done over "contract first" developed web services.

2.6 Mining Software Repositories (MSR)

Much of the software development nowadays is made by updating code in software repositories, mainly because version control systems can better administer code changes and keep developers up-to-date with modifications. Regarding this, many historical data are usually stored and increasingly used to perform some kind of research. Primarily, historical data inside software repositories were used to track defects, but several studies emerged from this, aiming to deal with other aspects in software development like software design, architecture and reuse (HASSAN, 2004). To retrieve and analyze historical data in software repositories, many solutions exists, like the ones compared in the study undertaken by Olatunji et al. (2010). Despite this, the industry do not take part in research regarding this kind of data, and so researchers are not aware of industrial practices (GLASS, 2003).

Kagdi et al. (2007) points that the expression "mining software repositories" derives from the fact that researchers use many approaches to extract proper information from repositories and detect trends in software evolution, activities that are similar to the ones related to data mining and knowledge discovery. These techniques can help developers in decision making processes to depend less on their personal experience and intuition and more on historical data analysis (HASSAN, 2004).

One of the many uses of MSR results is in the recommendation of practices and source code that might need to be modified based on the change history of repository files. Ying et al. (2004) shows an example of this need with a modification task of Mozilla Web Browser project. By suggesting relevant files to be modified due to changes in software code, MSR techniques can help to increase development speed and quality, creating systems more likely to be consistent and adaptive to evolutionary changes.

MSR is an evolving field of research and has numerous works published on the literature. Abate et al. (2015) presents a tool that mines components metadata to identify which components available at a repository cannot be installed on the main system, using three project repositories to evaluate the correctness of the tool. Regarding changes in applications, Ray et al. (2014) defines the concept of unique changes and provides a method for identifying them in software project history. Gala-Pérez et al. (2013) makes use of empirical metrics to check what can be inferred from them for projects of the Apache Software Foundation. To enable platform-independent software mining and metric visualization, Carvalho et al. (2015) presents a REST web service implementation directed to such tasks, in order to enable the development of service-oriented applications to mine and visualize software data across the web. Those are just some examples of different applications of MSR techniques.

2.6.1 MetricMiner

In order to proceed with MSR techniques over repositories, many solutions exists, like the ones presented by Peters and Zaidman (2012), Spacco et al. (2005) and Aniche et al. (2013). Here the focus will be on the latter, due to the fact that it can analyze source code changes by incorporating user customized investigation code.

Aniche et al. (2013) presents MetricMiner, a framework written in Java code that can be used to develop applications that perform deep analysis over GitHub repositories. It runs code written by its user over different software revisions in order to capture desired information for further analysis. Benefits from using MetricMiner relies on its multi-thread facility, which can improve complex analysis performance over repositories with huge amounts of files and/or revisions, and also by being highly customisable, as virtually any study can be designed to be performed over source code files. It must be pointed out that MetricMiner does not perform any kind of analysis over the retrieved information, which is a responsibility entirely on the researcher.

Listing 2.4 shows an example of how MetricMiner works[3]. The basic class `Study` has a single method called `execute`, that starts the mining procedure according to given parameters.

[3]Complete documentation and working examples are available in the official MetricMiner website at `http://www.metricminer.org.br/`.

Listing 2.4 - Implementation example of class `Study`

```
public class MyStudy implements Study {
  public static void main(String[] args) {
    new MetricMiner2().start(new MyStudy());
  }
  @Override
  public void execute() {
    new RepositoryMining()
    .in(<LIST OF PROJECTS>)
    .through(<COMMITS>)
    .process(<PROCESSOR>, <OUTPUT>)
    .mine();
    }
}
```

In the example above, `in` defines the GitHub projects that will be mined; `through` establishes the range of commits to be considered; `process` passes to MetricMiner the class that implements the metrics (the "processor") to be retrieved from the projects and the path where it should output the file with retrieved data; and `mine` starts the process. An example of a processor[4] is shown by Listing 2.5 (imports hidden).

Listing 2.5 - Example of processor implementation to be used by MetricMiner

```
public class DevelopersVisitor implements CommitVisitor {
  @Override
  public void process(SCMRepository repo, Commit commit, PersistenceMechanism writer) {
    writer.write(
      commit.getHash(),
      commit.getCommitter().getName()
    );
  }
  @Override
  public String name() {
    return "developers";
  }
}
```

It should be highlighted that the processor class is not different from any ordinary Java class. As so, it can define the procedures to be taken by MetricMiner inside it and/or use methods defined by other Java imported classes.

[4]Code snippets are available in the official MetricMiner website at `http://www.metricminer.org.br`.

3 PRELIMINARY STUDY

This chapter presents a starting study focused on some of the research questions listed in Chapter 1. Through the analysis of simple metrics applied over some projects repositories, it is verified how often changes occur in the number of basic XSD containers and the relation of such frequencies among distinct projects. This study was published in the 16th International Conference on Computational Science and Its Applications (ICCSA) in 2016, with the title *Evolution of XSD documents and their variability during project life cycle: a preliminary study* (ALMEIDA; GUERRA, 2016).

3.1 Research questions

In order to make inference about the behaviour of projects changes during their life cycles, some Preliminary Research Questions (PRQ) are defined, as follows:

- **PRQ1** – What is the frequency of changes in the number of each XSD basic container type (*element*, *attribute* and *complexType*)?
- **PRQ2** – Do distinct projects have similar changing frequencies for each XSD basic container type?
- **PRQ3** – What is the number of XSD document versions where no changes in the number of XSD basic container types are found?

3.2 Study methodology

To undergo this study, a short number of open-source projects hosted at GitHub needs to be selected, as MetricMiner only works on GitHub repositories. The selection criteria are as follows:

- Projects must have XSD documents in its repository;
- XSD documents must have enough modifications in the repository, represented by the number of commits of each XSD files, which is defined here as being at least five;
- Projects that defines or uses web services are desirable.

To evaluate projects, three simple metrics are defined, considering the number of three XSD basic container types: `element`, `attribute` and `complexType`. Using

these metrics, the quantity of modifications in XSD documents are mapped by their types (in elements, in attributes and in complex types). The documents used for this evaluation are obtained by historical mining over the selected projects repositories, in order to quantify each kind of change. Later in this dissertation, similar numbers, obtained from WSDL documents, will help to check if there are changing patterns in web services contracts during their development life cycles.

To achieve this first study proposal, the following information are retrieved from projects, in order to gather the desired metrics and to organize data:

- **HASH** – Hash number of commit;
- **REVCOMMIT** – Serialized number of commit, starting with 0 (most recent, or *"Head"*);
- **FILENAME** – Name of the analyzed file;
- **MODCOUNT** – Serialized number of modification index counter for current file, starting with 0 (most recent);
- **QTY_ELEMENTS** – Quantity of <xs:element> tags in current file;
- **QTY_ATTRIBUTES** – Quantity of <xs:attribute> tags in current file;
- **QTY_CTYPES** – Quantity of <xs:complexType> tags in current file;
- **MOD_ELEMENTS** – Direction of growth in <xs:element> tag quantity related to previous modification index, denoted by -1 for decrease, 0 for no change and 1 for increase;
- **MOD_ATTRIBUTES** – Same as above, but for <xs:attribute> tag;
- **MOD_CTYPES** – Same as above, but for <xs:complexType> tag;

In order to retrieve the aforementioned data, some code implementation is needed. To extract metrics from XSD files, a simple Java package named `XSDMiner` was written[1] and coupled to MetricMiner (ANICHE et al., 2013) into a processor for historical data mining over projects repositories. Running MetricMiner with `XSDMiner` outputs a Comma Separated Values (CSV) file with the desired information.

[1] `XSDMiner` is freely available as an Eclipse project at: `http://github.com/diegobenincasa/XSDMiner`

3.3 Implementation

To evaluate the pace of changes in XML documents used in web services contracts, this work uses the simple metrics defined in Section 3.2. Also, as said in the previous section, a Java package named `XSDMiner` was written to perform the study and retrieve metric data. The considered XSD containers at this study are counted for each XSD file found in projects repositories and compared between consecutive revisions (or *commits*) to check if there are evidences of a change.

`XSDMiner` implements the class `XSDParser`, which parses XSD files and retrieves the counting of container types listed before. It uses Java DOM to parse input files and has different methods for retrieving each container type counting. MetricMiner then runs over selected GitHub repositories (downloaded and available locally) and performs user-defined analysis (or "studies") at each revision. MetricMiner `Study` class was written so that it can use `XSDParser` to parse each XSD file in the repository, running its code for every commit. As MetricMiner outputs a CSV file, the `Study` class was written in such a way that information listed in Section 3.2 can be gathered using this file type. Listing 3.1 illustrates the implemented `Study` class to be used as a processor by MetricMiner (imports hidden from the code snippet). It shows a working example for a single repository (in this case, the one named "xwiki-platform") to extract metrics as required by the class `MineXSD` for every commit and using four threads at a time.

Listing 3.1 - XSDMiner code snippet: the `Study` class implementation

```
1  //...
2  public class MyStudy implements Study {
3
4      String projectDir = "/home/diego/github/";
5      String output = "/home/diego/Desktop/mm_output/" + project + ".csv";
6
7      public static void main(String[] args) {
8          new MetricMiner2().start(new MyStudy());
9      }
10     @Override
11     public void execute() {
12         new RepositoryMining()
13                 .in(GitRepository.allProjectsIn(projectDir))
14                 .through(Commits.all())
15                 .withThreads(4)
16                 .process(new MineXSD(), new CSVFile(output))
17                 .mine();
18     }
19 }
```

3.4 Study execution

Applying the criteria mentioned in Section 3.2 and using GitHub's native search mechanism, six projects were selected to be analyzed, as shown in Table 3.1.

Table 3.1 - Selected projects to perform the study

Project	Description	Address
Datacite/Schema	DataCite Metadata Schema	https://github.com/datacite/schema
OpenNMS	OpenNMS enterprise grade network management application platform	https://github.com/OpenNMS/opennms
SOCIETIES-Platform	SOCIETIES platform software	https://github.com/societies/SOCIETIES-Platform
spring-ws	Spring Web Services	https://github.com/spring-projects/spring-ws
XeroAPI-Schemas	XSD Schemas for api.xero.com	https://github.com/XeroAPI/XeroAPI-Schemas
xwiki-Platform	The XWiki platform	https://github.com/xwiki/xwiki-platform

After the projects were selected, their repositories were downloaded locally, and MetricMiner with `XSDMiner` was executed over each one of them. In this preliminary study, MetricMiner ran several times, each time for a single project, and individual CSV files were generated per project. To ease the analysis over those files, they were imported to a PostgreSQL database and a SQL script was written[2] to output metric data separated by XSD container type.

3.5 Results and analysis

Selected projects have different XSD file quantities, and some projects have a large set of files of that type. Thus, it is difficult to show complete collected data in a summarized manner. Table 3.2 presents raw metric information that could be gathered.

The chart presented in Figure 3.1 shows the percentage of commits where modifications were found, classified according to each container type (blue: `xs:element`; orange: `xs:attribute`; yellow: `xs:complexType`). On the other hand, chart illustrated in Figure 3.2 shows the ratio between the total number of modifications (separated by container type) and the number of commits where those modifications occurred (similar label as before), also classified according to each container type, which represents a mean value of container quantity changes

[2]The script is also available at `XSDMiner` repository at GitHub.

per commit with modifications.

Table 3.2 - Raw metric data extracted from projects

	Datacite/Schema	OpenNMS	SOCIETIES-Platform	spring-ws	XeroAPI-Schemas	xwiki-Platform
XSD files	44	100	78	53	35	4
Total modifications in elements	89	257	268	114	130	33
Total modifications in attributes	97	347	168	109	96	11
Total modifications in complex types	87	240	204	115	78	17
Total commits	200	43614	9761	4286	137	49187
Commits with XSD modifications	34	722	597	60	94	50
Commits with elements modifications	11	192	231	40	64	35
Commits with attributes modifications	19	339	93	36	31	9
Commits with complex types modifications	10	169	141	38	33	15
Commits with no modifications	166	42892	9164	4226	43	49137

Figure 3.1 - Percentage of commits where changes in each container type were found

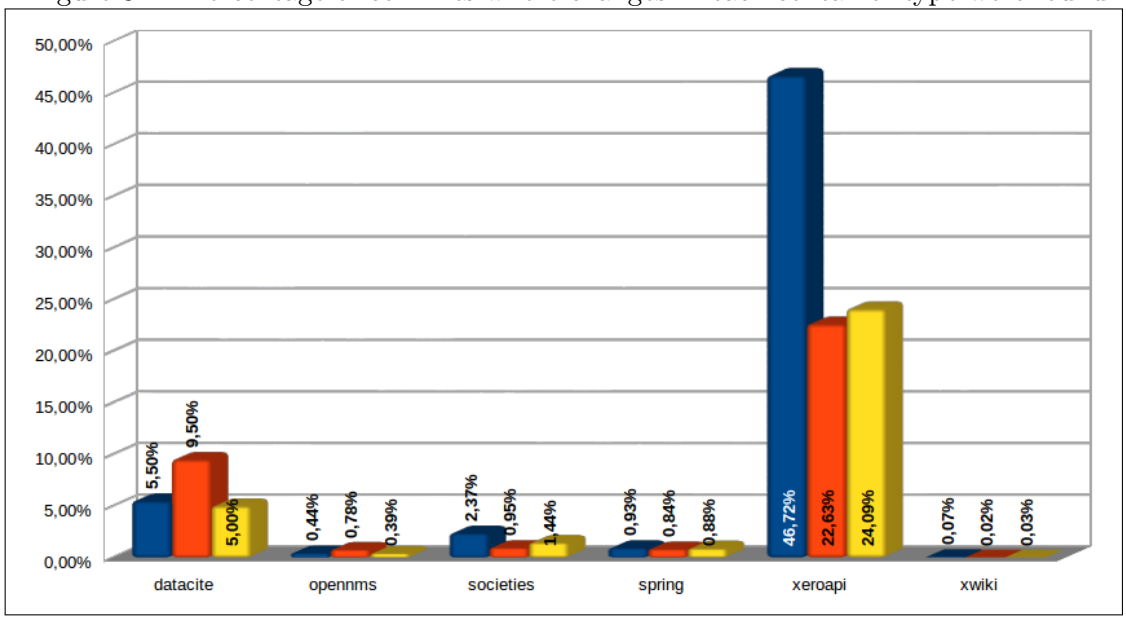


Figure 3.2 - Ratio between total number of each container type modification and the number of commits where those changes occurred

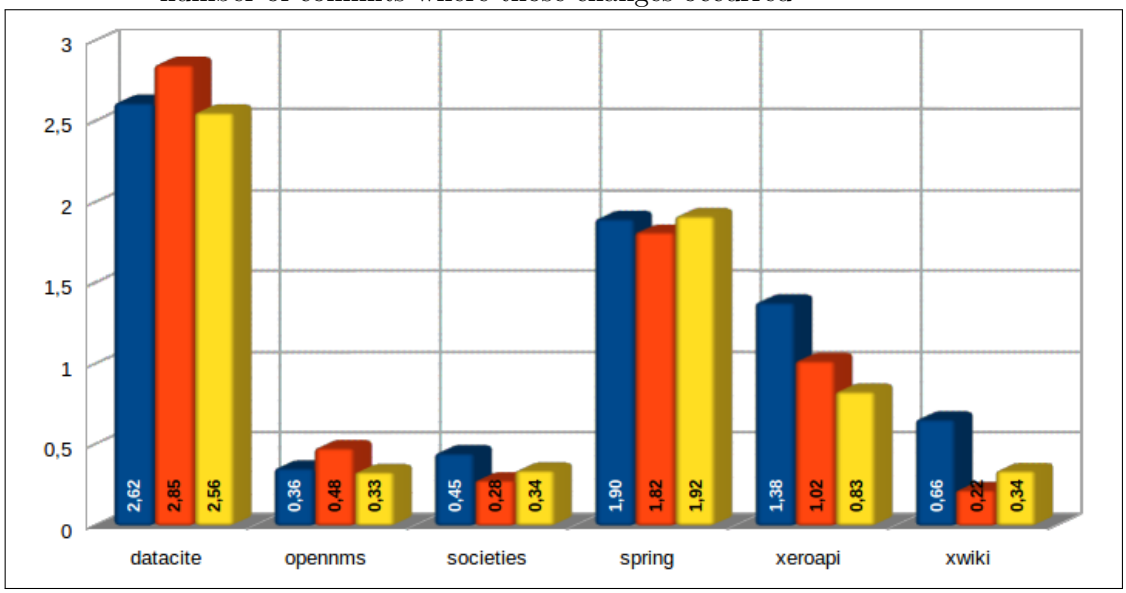


Figure 3.1 illustrates that there are no visible patterns regarding the percentage of commits with modifications. Project *spring-ws*, the only project in this study that deals with web services, shows similar numbers for all container types, very much alike other three projects, but distinct from the other two. Four of the six selected projects have greater percentages in `element` containers, and they show a similar tendency: `element` containers have greater percentages, followed by `complexType` and `attribute` containers (in this order). The other two projects also seem to share a common behavior: greater percentages for `attribute` containers, followed by `element` and complexType containers (in this order). One project stands out from the behavior of the others: `XeroAPI-Schemas` changes the number of every container type in a frequency much larger than other projects, particularly the `element` container. Not with the same highlight, but also with a much different behavior in respect to the other projects, is `Datacite/Schema`, which shows a larger number of commits with modifications in `attribute` containers in comparison to other types.

The chart illustrated in Figure 3.2 shows that, visually, no pattern can be inferred, in a similar conclusion as the one found by inspecting Figure 3.1. However, the calculated ratios seem to approximate numbers for the three considered container types when analyzing projects individually. Nevertheless, there is not even a "container type ranking" pattern as in Figure 3.1, with induces the need to deepen the study and consider more variables and metrics in the analysis to reduce the

seemingly heterogeneous behaviour of ratios.

No matter how often modifications occur inside XSD documents, they reveal another common characteristic: increase or decrease of containers do not prevail one over another. This establish a new issue to address later in this dissertation, as those two metrics can not be used to sustain an argument for tendencies on XSD documents modifications. Examples of such heterogeneity are shown in Figure 3.3, Figure 3.4 and Figure 3.5. Blue lines represent `element` containers; red lines represent `attribute` containers; and yellow lines represent `complexType` containers. Left-to-right reading of charts goes from older to recent commits.

Figure 3.3 - Quantities of containers through XSD modification commits for project `OpenNMS`, file `users.xsd`

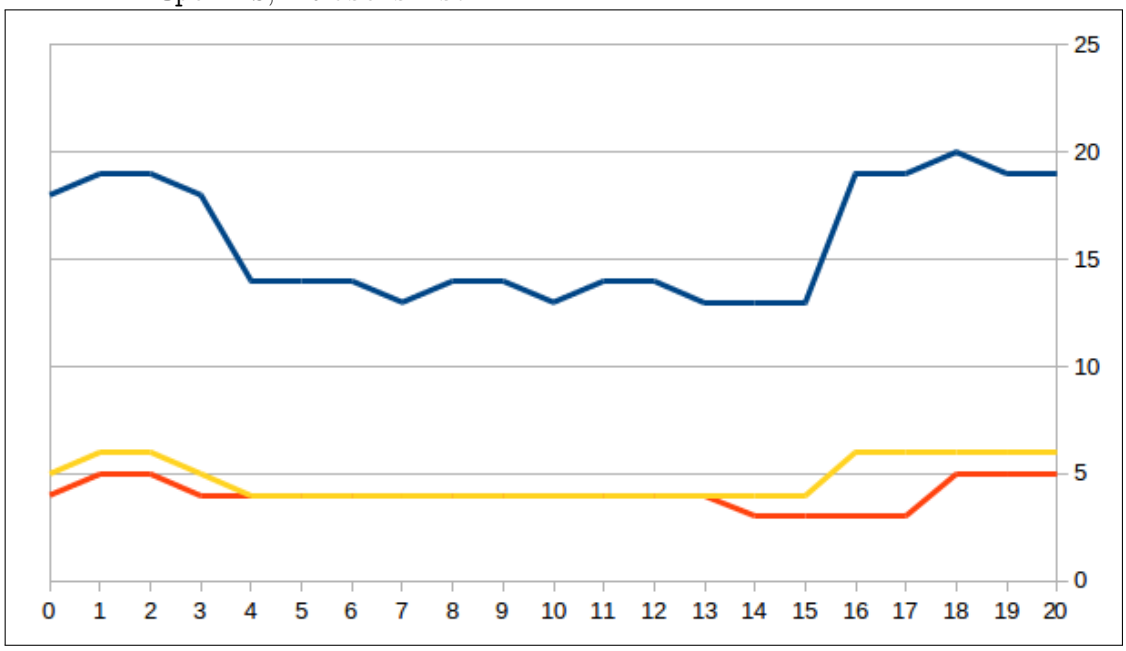


Figure 3.4 - Quantities of containers through XSD modification commits for project `SOCIETIES-Platform`, file `org.societies.api.internal.schema.privacytrust.privacyprotection.model.privacypolicy.xsd`

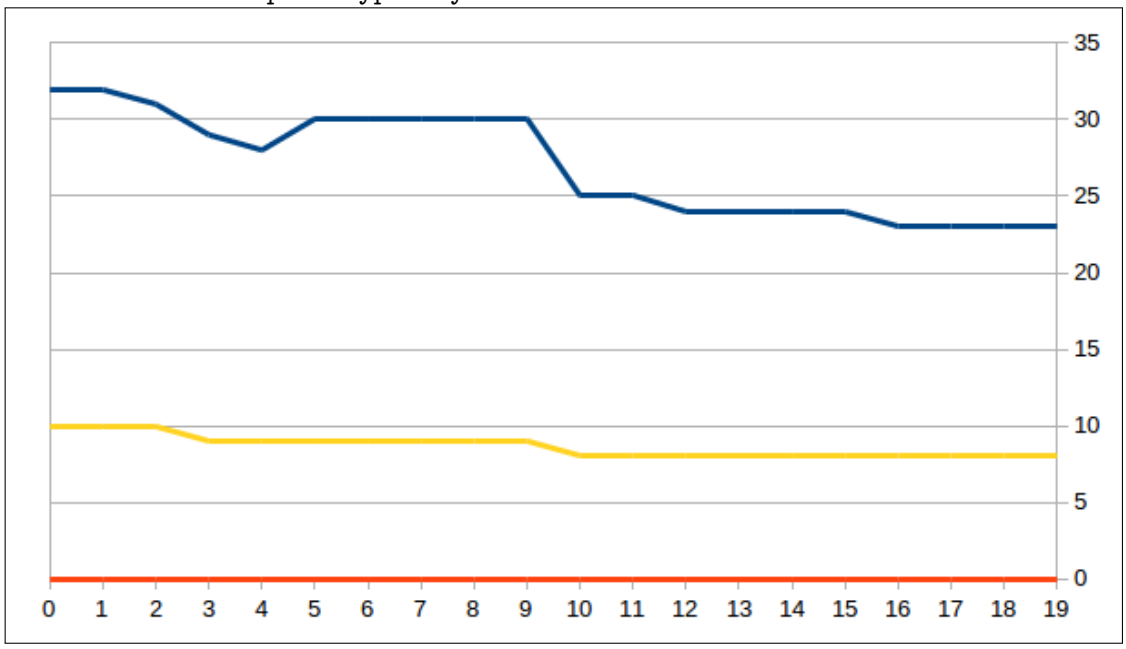
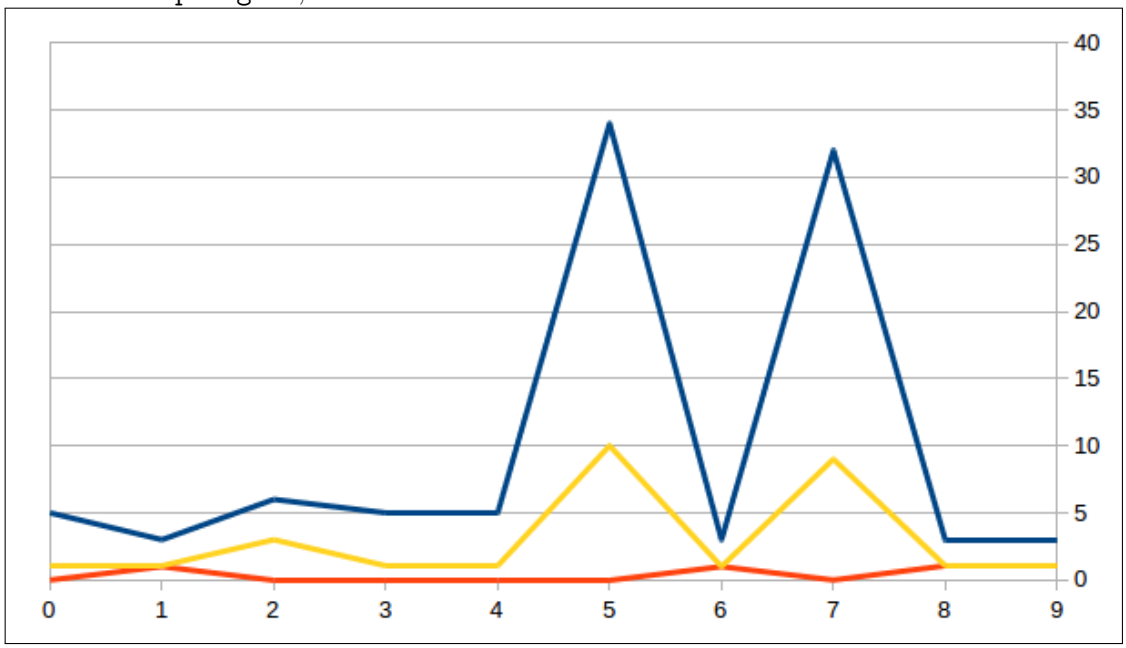


Figure 3.5 - Quantities of containers through XSD modification commits for project `spring-ws`, file `schema.xsd`



3.5.1 Answering the research questions

(PRQ1) – What is the frequency of changes in the number of each XSD basic container type (*element*, *attribute* and *complexType*)? – The occurrence frequency of changes varies a lot per project and per XSD file. While some projects have more modifications in "element" containers, other ones show this behavior in "complex types". Besides that, different files have distinct modification behaviors, which hampers the task to establish a fixed changing rate for each modification type. Nevertheless, apparently elements modifications are the most frequent and happens several times during projects life cycle.

(PRQ2) – Do distinct projects have similar changing frequencies for each XSD basic container type? – Selected projects do not fully share a common behavior regarding changing frequencies, but two groups can be defined: one consisting of four projects in which frequencies of changes follow the sequence `attribute`, `complexType` and `element` (sorted ascending); and another consisting of two projects in which frequencies of changes follow the sequence `complexType`, `element` and `attribute` (sorted ascending). The number of projects selected do not enable the establishment of a unique answer, and an increase in this number might lead to one.

(PRQ3) – What is the number of XSD document versions where no changes in the number of XSD basic container types are found? – Three of the six selected projects present changings in some container type quantities in a frequency not greater than 1% of total commits. Other two projects shows a bit higher values, but do not reach 10% of total commits, and one project presents results in complete disparity with the others. This reveals that projects that defines XSD documents do not alter these files in a regular basis. On the other hand, they perform many modifications during their life cycles (as seen on Table 3.2), which sustains the need to deepen the study to understand how and where the changes in XSD files occur, as well as to analyze the impact on the contract semantics.

3.6 Threats to validity

The number of projects selected by the criteria specified at Section 3.2 is small when compared to the many existing open-source projects that deals with web services. The major problem with the selection lies on GitHub search engine: it does not provide an inside-repository search tool that could retrieve projects with specific files in it – like any XSD file (`*.xsd` filter, grouped by project), for example. This makes

the search and selection of projects even more difficult. To improve the quality of the analysis undertaken in this study, it should be continuously executed over other projects, and different ways to search for relevant ones should be considered.

Another fact that should be taken into account is that some modifications might become masked by other ones. As an example, in a certain revision a container might have been created, but this can cover up the removal of another container of the same type, as the current study is mainly based on the number of containers found at each revision of XSD documents. This problem needs to be addressed, analyzing each modification individually.

Projects have different life cycles, and so have distinct revision quantities and, ultimately, diverse modification rates. This fact is not addressed in the present study, as it only considers the absolute values of the extracted metrics and its percentage portion in total modification count. It does not relate those numbers with the project life cycle, which could hide knowledge not retrieved with this work.

3.7 Conclusions

This preliminary study aimed to check if changes occur in XSD schemas, as well as to understand how these modifications happen among consecutive documents revisions. Research found that the frequency of changes are not homogeneous for the projects analyzed, and the types of modifications also do not share a common changing behavior. Thus, it was unable to establish a changing pattern for all six projects. Also, the number of projects studied prevents the generalization of the results for any project that defines XSD contracts. Research questions were answered with no final statements, as results per project and per modification type are not entirely similar.

On the other hand, a relevant information obtained was that contracts do experience many modifications, which is natural but can impact on consumers of data that relies on these schemas. Thus, further analysis with a larger amount of projects need to be undertaken to unveil their particularities, addressing each modification individually and verifying what was changed at each commit. This was left to be done in the sequence of this dissertation. It must be stated that this next research needs to be done with web service projects only.

4 RESEARCH DESIGN AND EXECUTION

After the preliminary study undertaken in Chapter 3, next step is to perform a wider study to investigate contract changes deeper. It involves dealing with modifications individually for each XSD schema inside or outside WSDL documents and for each project. This chapter presents the design and execution of a study that aims to analyze those modifications and identify common characteristics among projects based on changing tendency of their WSDL contracts.

A caveat to define patterns or tendencies in the preliminary study was the number of projects analyzed, which brings difficulties to generalize the results obtained. Thus, in this next step, the amount of projects was the first point to consider before any further implementation and analysis. Nevertheless, it also includes more metrics to address changes not dealt before: the preliminary study only took account of each tag type through revisions, and now changes are analyzed individually by modification type.

The previous study concluded that changes occur frequently in XSD documents, which might cause semantic changing and impact on provided data interpretation by service consumers. The frequency of changes, though, needs to be more investigated. Moreover, the existence of service-specific contracts defined as WSDL documents in the repository has to be the main requirement to select projects, and those files should be analyzed alongside with dedicated XSD documents (if they also exist). Due to this, next steps aim to address those issues and answer some research questions (RQ).

4.1 Research questions

To achieve the goal of the present study, the follwing RQs are defined:

- **RQ1** – What is the occurrence rate of each XSD modification type?
- **RQ2** – With which pace XSD modifications include or exclude information?
- **RQ3** – How is the distribution of modifications among commits?

Answers to these RQ's should give relevant knowledge to the understanding of the behavior of contracts modifications.

4.2 Study methodology

Although the number of selected projects led to some valuable inferences on the study undertaken at Chapter 3, it is not adequate to establish reliable conclusions that can be generalized. As so, new approaches to select appropriate projects must be considered.

One of the greatest difficulties on selecting a relevant number of projects relies on the fact that GitHub search tool does not provide means to search for projects with the desired characteristics for this dissertation (same as listed at Section 3.2). Thus, at the time of the preliminary study, only few projects were chosen.

Google has mirrored GitHub database to their own big data query solution, called Google BigQuery [1], which is a more suitable tool to search for projects with the desired characteristics. Being, according to Google, a "fully managed, petabyte scale, low cost analytics data warehouse", the benefits regarding GitHub public dataset at Google BigQuery include: a real database structure for every project, every file and all their revisions, with many attributes stored as table columns; an easy-to-use query interface, that queries the database with a SQL-like syntax and outputs data that can be exported to different file formats (CSV, JSON, etc.); a fast and reliable mechanism to rapidly retrieve queried data. By using Google BigQuery, a greater amount of projects were selected to create a representative sample for the results to be obtained in this research.

Having projects with WSDL documents in their repository is not enough, as they might have a few number of revisions for those files, which is not adequate for a representative analysis. After selected projects being checked out locally, they were filtered according to a minimum number of desired revisions for at least one WSDL file. For this research, the minimum number of revisions was defined as five.

During the selection phase, some WSDL files were found with syntax errors, which could muddle the analysis algorithm and break the software execution. To prevent this problem, all documents were validated before starting the analysis and, in case of problems, the problematic files were removed from the processing and no revisions of them were considered. If these procedures made projects not have at least one WSDL file with five or more revisions, those were entirely removed from the analysis.

After having a projects foundation to work with, the metrics considered at

[1] Available at: `https://cloud.google.com/bigquery/`

Section 3.2 should be expanded. From that moment on, those metrics were defined as follows:

- Quantity of `xs:element`, `xs:attribute`, `xs:complexType` and `xs:import` tags;
- Increase and decrease of each tag listed above after each commit;
- Quantity of tags refactored (name changed), except for `xs:complexType`;
- Quantity of tags relocated inside schema, for `xs:element` and `xs:attribute`;
- Quantity of projects with at least one WSDL document with more than five revisions;

The metrics defined before should be extracted from each WSDL/XSD file revision, but only from files that meet the requirement of having at least five revisions. Once extracted, it is desired that they can lead to answering the research questions listed in Section 4.1. From the list of metrics listed before, it can be seen that the first one was already implemented and used for the preliminary study.

To gather the metrics defined above, a new MetricMiner `Study` class was written, in order to extract the XSD schema from within the WSDL documents for further calculations. To ease these calculations and the analysis to be proceeded after that, every schema extracted by MetricMiner over the entire range of commits were imported to a PostgreSQL database into a table with some information, namely:

- Schema count (a single WSDL document can have multiple defined XSD schemas);
- Name of the WSDL (or XSD) file associated with the schema;
- Name of the corresponding project;
- Commit hash code of the file;
- Timestamp of the commit;
- The schema code itself;

- A flag “a” (add) or “r” (remove) to identify a commit where the file appears in or is removed from the repository;

Once the database was populated with all the schemas from the included projects, another Java code was written to compare each pair of consecutive schema revisions, extracting the defined metrics. The result were then inserted into another database table in order to be later exported and analyzed.

Final steps were related to reviewing the analysis and trying to answer the RQ’s already defined. Pie charts, histograms and statistical data over the extracted metrics were generated trying to find common behaviours of the modifications that could answer these questions.

4.3 Projects selection

As said in Section 4.2, Google BigQuery database were used for projects selection, as it has more relevant tools to filter GitHub projects according to this research needs. By using its interface for public dataset `github_repos`[2], the following query was executed:

Listing 4.1 - Google BigQuery query string to select projects with WSDL documents in their repository

```
SELECT commit, committer.date AS date, repo_name, difference.new_path AS file
FROM FLATTEN([bigquery-public-data:github_repos.commits], difference)
WHERE LOWER(RIGHT(difference.new_path, 5)) = ".wsdl"
AND difference.old_sha1 <> difference.new_sha1
```

The above query is looking for commits from any repository where the involved file ends with `".wsdl"` extension. It must return the commit hash code, the commit date, the repository name and the file name and relative path.

The above query returned[3] a list of files with their commit hash and date, as well as their repository names. This list was then exported as a CSV file and imported to a local PostgreSQL database. Then, a simple SQL query listed distinct project repository names in it, leading to a count of **210** GitHub projects that have at least one WSDL document in their repository. The minimum revisions quantity threshold filtering was done in a next step, after checking out every project listed.

During the checkout of projects, it was noticed that Google BigQuery also returns

[2]Available at: https://goo.gl/o9Aeuz.
[3]Last queried on September 13th, 2016.

all forks for the same project. This can be seen by looking at the repository names, where the first part (before "/" character) is the name of the fork maintainer and the second part (after "/" character) is the actual name of the project (for example, a repository named "wso2/carbon-data" is related to project "carbon-data", under responsibility of GitHub user "wso2"). Due to that, another filter was then defined: the main project should be considered without any forks. This ensures that only one edition of each project is analyzed, preventing multiple interpretations of the final results. After applying this filter, using a query to split the text after "/" in the repository name and listing distinct resulting texts, **187** projects remained at the list. It must be stated that when a project was spotted having forks, the discovery of the main one was done manually and its repository address was kept in a separate list for checkout.

With a primary list of projects established, next step was to check which files have at least five revisions in their repository, verifying how many projects still obey the selection criteria. This was done by querying the projects database table with a statement to filter tuples with more than five occurrences for the same file and list their project names. This led to a final count of **159** projects.

A limitation of Google BigQuery is that it does not include information about the branch of the files hosted at their servers. Thus, when checking out projects, it was defined that only the branch `master` should be considered.

4.4 Metrics and analysis implementation

To proceed with the analysis, new Java classes needs to be implemented, in order to extract useful data from WSDL and XSD documents and to compare consecutive revisions. Thus, two classes were written for those tasks: `XSDMiner2`[4], to extract XSD schemas from within WSDL documents (when it is the case) and commit them to a PostgreSQL database table, and `SchemaCompare`[5], to check each pair of consecutive revisions of schemas and verify the modifications, committing them to another database table.

`XSDMiner2`, like `XSDMiner`, is a extension of the `Study` class from MetricMiner. It iterates over each commit, checking for the existence of a WSDL/XSD document at it and, in case of existence, extracts only the XSD schemas (via an auxiliary class `XSDExtractor`) and commits to a database table along with some data, as listed in

[4] Available at: http://www.github.com/diegobenincasa/XSDMiner2.
[5] Available at: http://www.github.com/diegobenincasa/SchemaCompare.

Section 4.2. A code snippet from `XSDMiner2` is presented in Listing 4.2.

Listing 4.2 - `XSDMiner2` code snippet, with some parts excluded

```
for(Modification m : commit.getModifications())
{
//...
    String fileExtension = fName.substring(fName.lastIndexOf(".")+1);

    boolean isWSDL = fileExtension.equals("wsdl");
    boolean isXSD = fileExtension.equals("xsd");

//...
    String[] schemas;
    if(fileExtension.equals("wsdl"))
    {
      XSDExtractor.wsdlToXSD(input, outputXSD);
      schemas = XSDExtractor.splitXSD(outputXSD.toString());
    }
    else
    {
      outputXSD.write(IOUtils.toString(input).getBytes());
      schemas = new String[1];
      schemas[0] = outputXSD.toString();
    }

//...
}
```

`SchemaCompare`, on the other hand, do not use MetricMiner anymore. It just iterates over each schema stored at the database and compares them consecutively, from the oldest to the newest revision. Using the `XMLUnit` library [6], which is a tool that compares two XML documents and lists their divergences, it collects each difference found between revisions and stores the involved XML containers (with metadata) into Java hash maps, in order to analyze if the change type is one of these: container refactored to a different name; addition or removal of container; container relocation inside schema. It must be stated that this study deals with four types of XML containers: `element`, `attribute`, `complexType` and `import` tags. Class implementation is shown as a code snippet at Listing 4.3.

Listing 4.3 - `SchemaCompare` code snippet, with some parts excluded

```
//...
    XMLUnitCompare comp = new XMLUnitCompare();
//...
  for(String p : projects)
  {
//...
```

[6] Available at: `http://www.xmlunit.org/`.

```
7         for(String fn : files.keySet())
8       {
9  //...
10            for(int i = 1; i <= nSchemas; i++)
11        {
12 //...
13            while(rs.next())
14            {
15              if(counter > 0)
16              {
17 //...
18                          if(baseSchema.isEmpty() && !testSchema.isEmpty())
19                    comp.init(testSchema, 2);
20                  else if(!baseSchema.isEmpty() && testSchema.isEmpty())
21                    comp.init(baseSchema, 1);
22                  else if(baseSchema.isEmpty() && testSchema.isEmpty())
23                    comp.init(baseSchema, 3);
24                  else
25                    comp.init(baseSchema, testSchema);
26 //...
27              }
28 //...
29            }
30        }
31      }
32 //...
33   }
```

`SchemaCompare` class compares two consecutive file revisions and lists their differences regarding the tags defined to be studied. The data it generates is committed to a PostgreSQL database table to ease the analysis and the presentation of useful knowledge. This table has the following structure:

- Commit timestamp of the test schema (the one in the newest file revision in the comparison);
- The project name of the compared files;
- Commit hash codes of the base schema (the one in the oldest file revision in the comparison) and of the test schema;
- The name of the files where the compared schemas belong;
- The schema identifier (WSDL documents can have more than one schema defined inside them);
- Indicators of modification occurrence as numbers (0 for no changes between revisions, 1 for any change), one column for each modification type (as listed in Section 4.2);

- Quantities of each tag type in both schema revisions, one column per tag, and each tag divided into "before" (base schema) and "after" (test schema) quantities.

Refactoring, in the scope of this study, is defined as a tag name being modified. An analysis of tags between two file revisions, in order to infer that two are the same but with distinct names and properties, is complex and demands a deeper study not addressed here. So, in this research, two tags are considered the same when their names match over 90% by normalized Levenshtein distance. This was implemented in `SchemaCompare` class using the external Java library `java-string-similarity`[7].

Each difference found at the comparisons made by `SchemaCompare` class was also stored into the database. Further analysis was done by writing SQL scripts suited to generate useful information that can lead to answering the research questions listed at Section 4.1. Those scripts were explained during data extraction at the next section.

4.5 Data extraction execution

After classes being implemented, they were executed to gather data to be analyzed. Before the analysis, however, extracted schemas were validated in terms of formatting to prevent misinterpretation due to code malfunction. After this, a series of graphical and statistical analysis were undertaken to try to answer the research questions. All those steps are detailed in this section.

4.5.1 Data validation

As said in Section 2.2, XSD schemas inside WSDL documents should be inside `<types></types>` tags. As the name suggests, it describes the data types used for service messages. During the schema extraction using `XSDMiner2`, it was noticed that some WSDL files do not have those types definitions, not even referenced from an external XSD document, and so could not be used for any kind of analysis. This is probably due to the need of the involved projects to include "test files", which do not really represent a real service contract and are used just for project compilation and testing. Those files, then, were excluded from the study, to prevent bad interpretation of the results.

`SchemaCompare` class has an useful and important feature: it validates a XSD schema

[7] Available, with documentation, at: https://github.com/tdebatty/java-string-similarity.

according to the XSD standard. The `DiffBuilder` class of XMLUnit, which is used by `SchemaCompare` and is the main responsible to gather all differences between two XML documents, crashes when any of the involved files is malformed. During the processing of `SchemaCompare` over selected projects, some crashes occurred and, to prevent bad inferences about the projects that led to failures, those were excluded from the analysis.

A single WSDL document can define multiple schemas inside it. This possibility was addressed in `XSDMiner2`, and it stores the schemas with a "schema identifier" (integer) in the database. This, however, brings a problem: a single WSDL file can have more than five revisions, but inner schemas can have less than that individually. This is not an issue, as `SchemaCompare` considers the total number of revisions of the file itself.

A summary of `XSDMiner2` and `SchemaCompare` operations for projects validation is presented in Table 4.1 and in Table 4.2. As a consequence of the problems stated before, twenty projects were excluded from the projects listing obtained at Section 4.3, leading to a final number of 139 projects. All filterings and the final count of projects are summarized in Figure 4.1.

Table 4.1 - `XSDMiner2` and `SchemaCompare` filtering summary for all projects

File type	File status	Quantity	Percentage
WSDL	Valid	1800	97.72%
	Invalid	40	2.17%
	Excluded	2	0.11%
	Total	1842	
XSD	Valid	3617	99.34%
	Invalid	24	0.66%
	Excluded	0	0.00%
	Total	3641	
	Total files revisions	32787	

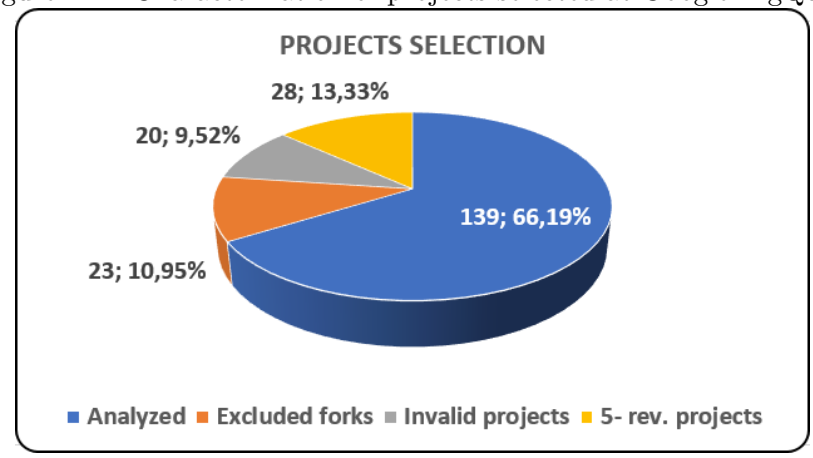
The final count of 139 projects to be analyzed were considered a representative sample, as it theoretically includes 66.19% of all GitHub projects that defines WSDL contracts and XSD schemas[8].

[8] As for September 13th, 2016.

Table 4.2 - Step-by-step projects filtering after Google BigQuery selection

Step	Description	Quantity	Percentage
1	First selection total	210	100%
2	Excluded forks	23	10.95 %
3	Projects with less than 5 revisions	28	13.33%
4	Excluded projects	20	9.52%
5	Final list to be analyzed	139	66.19%

Figure 4.1 - Characterization of projects selected at Google BigQuery



4.5.2 Projects profiles

Not all WSDL/XSD documents found have a minimum of five revisions, and the number of files, per project, that meets this requirement differs among projects. This is made clear with the graph in Figure 4.2. It shows the percentage of projects with less than a specific amount of WSDL/XSD documents with at least five revisions. It is noticed that a huge amount of projects (97.2%, or 139 of 143 projects – excluded projects were taken into account) have less than 20 contracts with more than five revisions, which can present a tendency of web service projects to not modify too much contracts in general. Data for this graph is listed in Table 4.3, which shows that one single project has 387 documents that meet the five revisions requirement (it is not represented in the graph), which is completely out of the general tendency.

Figure 4.2 - Percentage of projects with respect to the amount of WSDL/XSD contracts with more than five revisions

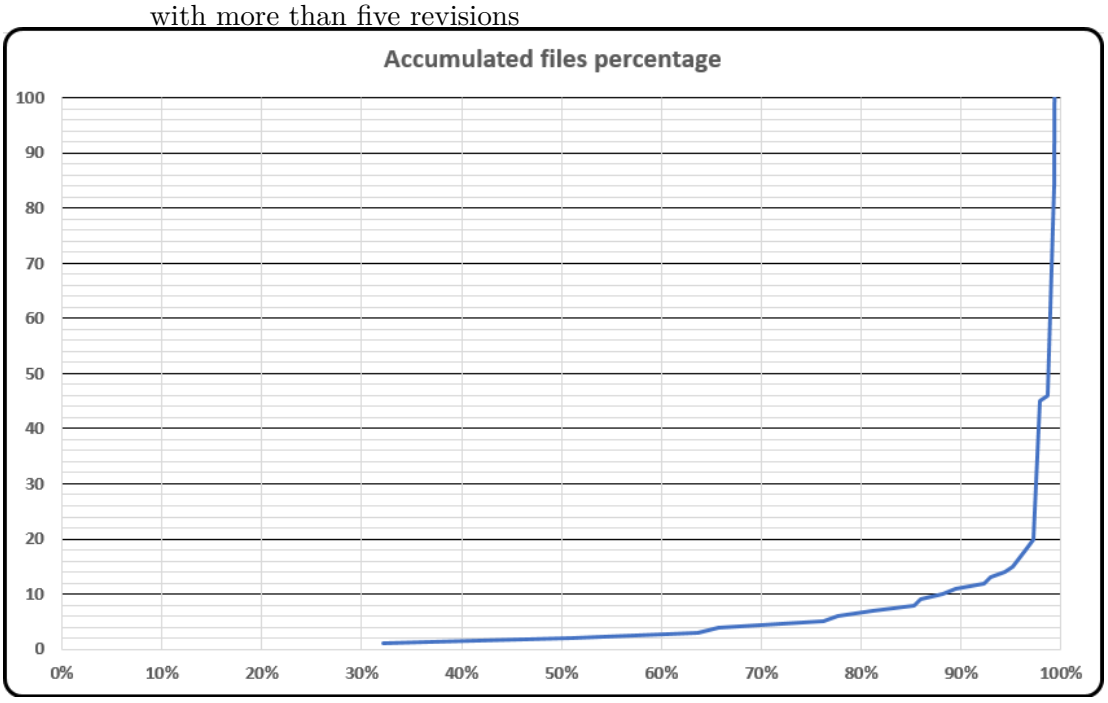


Table 4.3 - Percentage of projects with respect to the amount of WSDL/XSD contracts with more than five revisions: data table

Files	Projects	Percentage	Accum. Percentage	Files	Projects	Percentage	Accum. Percentage
1	46	32.17%	32.17%	12	4	2.80%	92.31%
2	27	18.88%	51.05%	13	1	0.70%	93.01%
3	18	12.59%	63.64%	14	2	1.40%	94.41%
4	3	2.10%	65.73%	15	1	0.70%	95.10%
5	15	10.49%	76.22%	18	2	1.40%	96.50%
6	2	1.40%	77.62%	20	1	0.70%	97.20%
7	5	3.50%	81.12%	45	1	0.70%	97.90%
8	6	4.20%	85.31%	46	1	0.70%	98.60%
9	1	0.70%	86.01%	84	1	0.70%	99.30%
10	3	2.10%	88.11%	387	1	0.70%	100.00%
11	2	1.40%	89.51%				

Some projects have all their contracts with more than five revisions, others have no documents satisfying this criteria, and many have variable numbers for each situation. As can be seen in Figure 4.3, projects usually have more files with less than five revisions. It was verified that only 24.5% of analyzed projects have most

documents with five or more revisions. This shows that, usually, modifications are less frequent per contract and more distributed among files.

The inference made before can be reinforced by Figure 4.4. For each project, it was calculated the percentage of WSDL/XSD documents that have more than five revisions in respect to the total amount of those documents. Data from all projects were, then, classified in four percentage quartiles, and it was verified that almost 50% of projects have less than 25% of their contracts satisfying the revision quantity threshold.

Figure 4.3 - Contracts with more than five revisions with respect to the ones that do not satisfy this criteria

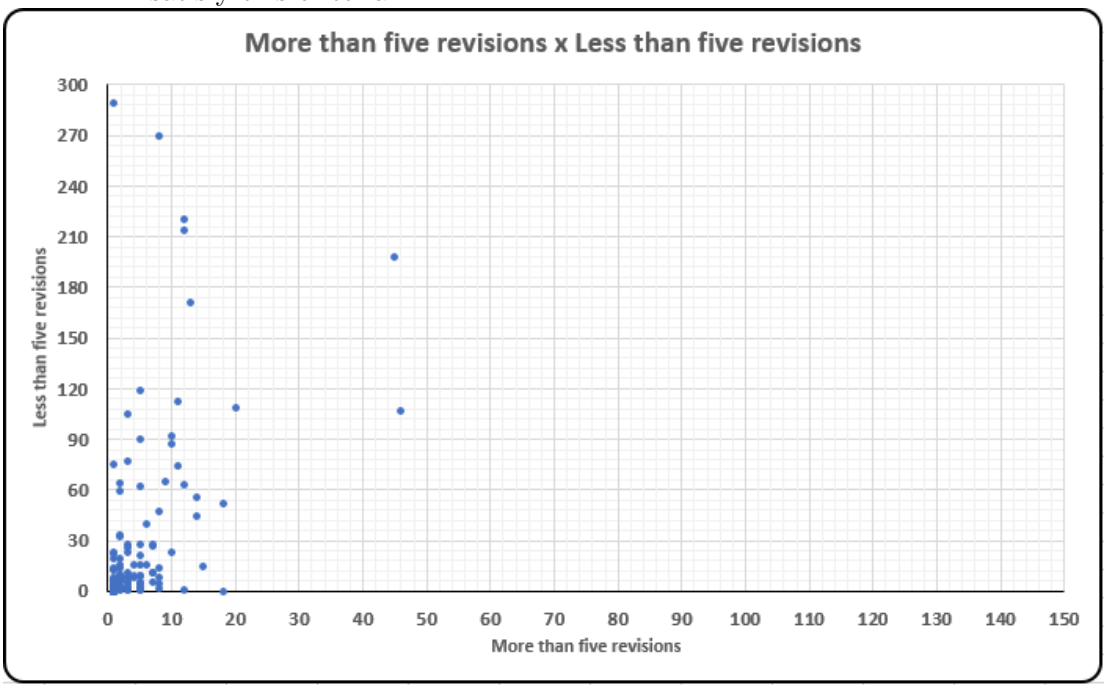
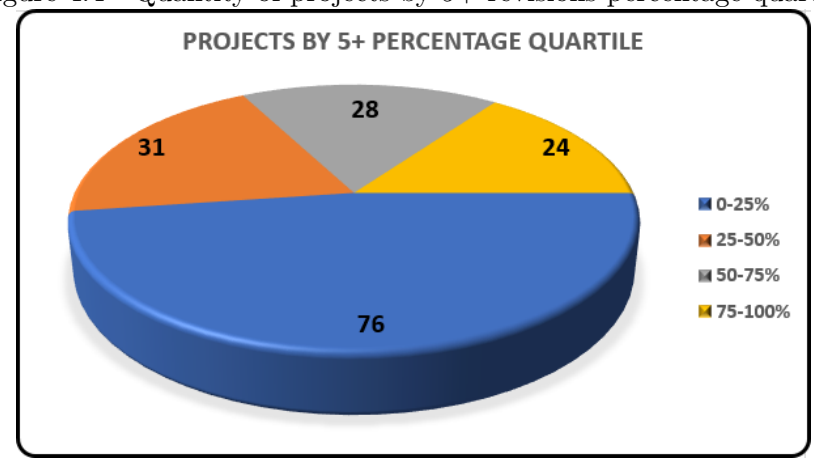


Figure 4.4 - Quantity of projects by 5+ revisions percentage quartiles



4.5.3 Contracts changes analysis

The implemented `XSDMiner2` class outputs several relevant data, which are then imported to the PostgreSQL database to ease the analysis. Such data are, basically, the amount of each XML schema tags considered in this study, as listed in Section 4.2. With those amounts for each file and their revisions, class `SchemaCompare` is executed to update the database with the additions, removals, relocations and refactorings of tags. This is a deeper analysis in comparison to what was made in Chapter 3, because while the preliminary study only considered the amounts of each tag per file revision, the present one also verifies each change that the documents experienced among commits. This unmasks changes not detected by simple tag quantities verification – for example, when a number of a particular type of tag is removed at the same time an equal amount of it is added. A summary of these types of changes is presented in Table 4.4.

Table 4.4 - Analyzed changes

Modification type	Description
Addition	Tag appears where it was not present in previous revision
Removal	Tag disappears where it was present in previous revision
Relocation	Tag disappears where it was present in previous revision and appears at a different place
Refactoring	Tag changes its “name” property by less than 10% (normalized Levenshtein distance)

Detection of changes listed in Table 4.4 were made using `XMLUnit` library. Its

`DiffBuilder` class gets two XML sources as input, and outputs all their differences as a list containing the compared nodes from each source and the difference itself. `SchemaCompare` class, then, uses this list to grab the types of the compared nodes and their `name` property to check what type of modification occurred. In case of any of the types listed in Table 4.4 were detected, the counting of it was computed. Counting modifications separately by change type prevents an addition-removal masking. A code snippet from `DiffBuilder` class is presented in Listing 4.4.

Listing 4.4 - `DiffBuilder` class from `XMLUnit` code snippet

```
//...
public void listAddAndRemove()
{
    Diff myDiff = DiffBuilder.compare(Input.fromString(baseSchema))
                .withTest(Input.fromString(testSchema))
                .withNodeMatcher(new DefaultNodeMatcher(ElementSelectors.byNameAndAllAttributes))
                .ignoreComments()
                .ignoreWhitespace()
                .checkForSimilar()
                .build();

    Iterable<Difference> diffs = myDiff.getDifferences();
    NormalizedLevenshtein nl = new NormalizedLevenshtein();
    Map<String, String> addedTags = new HashMap<String, String>();
    Map<String, String> removedTags = new HashMap<String, String>();
    Map<String, String> refactoredTags = new HashMap<String, String>();
    Map<String, String> relocatedTags = new HashMap<String, String>();

    if(myDiff.hasDifferences())
    {
      for (Difference difference : diffs)
      {
//...
```

A summary of data after this update is presented in Figure 4.5 and Table 4.5, as well as in Figure 4.6 (in absolute numbers). The charts illustrate that the modifications are most related to addition and removal of tags, and in a much higher amount for `xs:element` tags. It also shows that `xs:import` tags are too less modified, and that `xs:element` and `xs:complexTypes` experiences the greater amounts of changes found. Figure 4.5 also presents that the proportions of each modification type are similar between `xs:element` and `xs:attribute` tags. Details for each project are listed in Appendix A.

Checking the charts, it is verified that `xs:complexType` tags do not experience refactorings. This is due to the fact that `xs:complexType` do not usually have the "name" property that was analyzed for name changing. Due to the same reason, it was not possible, by the techniques defined for this research, to verify if tags of this

Table 4.5 - `SchemaCompare` output summary

Modification type	`xs:element`	`xs:attribute`	`xs:complexType`	`xs:import`
Addition	10764	327	4254	9
Removal	8125	340	1957	0
Relocation	4407	133	-	-
Refactoring	533	15	0	-

Figure 4.5 - `SchemaCompare` output summary

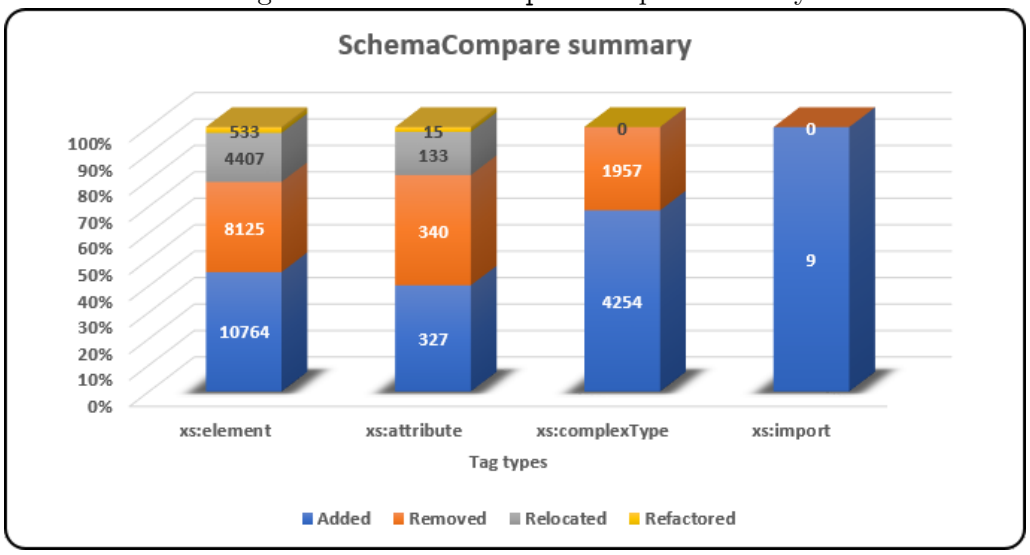
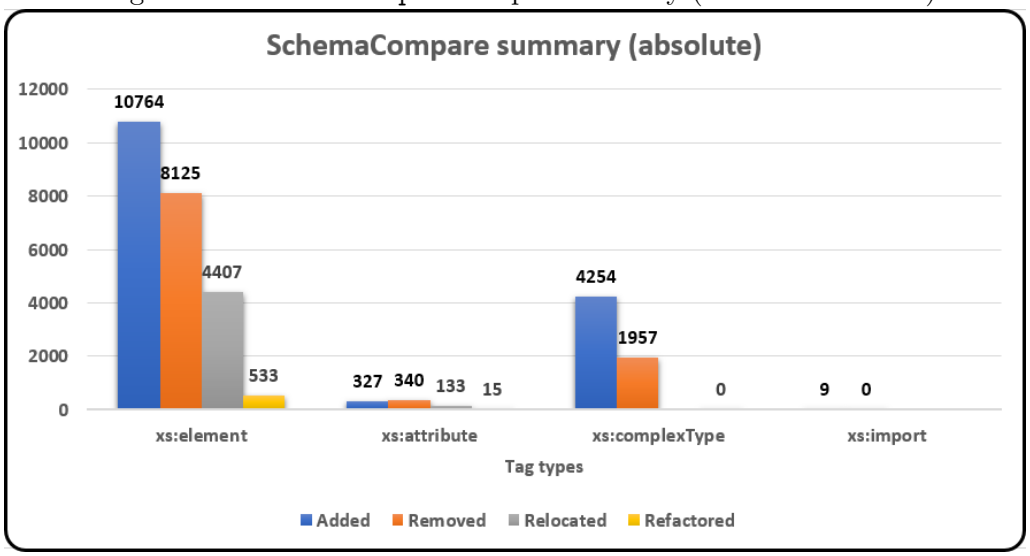


Figure 4.6 - `SchemaCompare` output summary (absolute numbers)

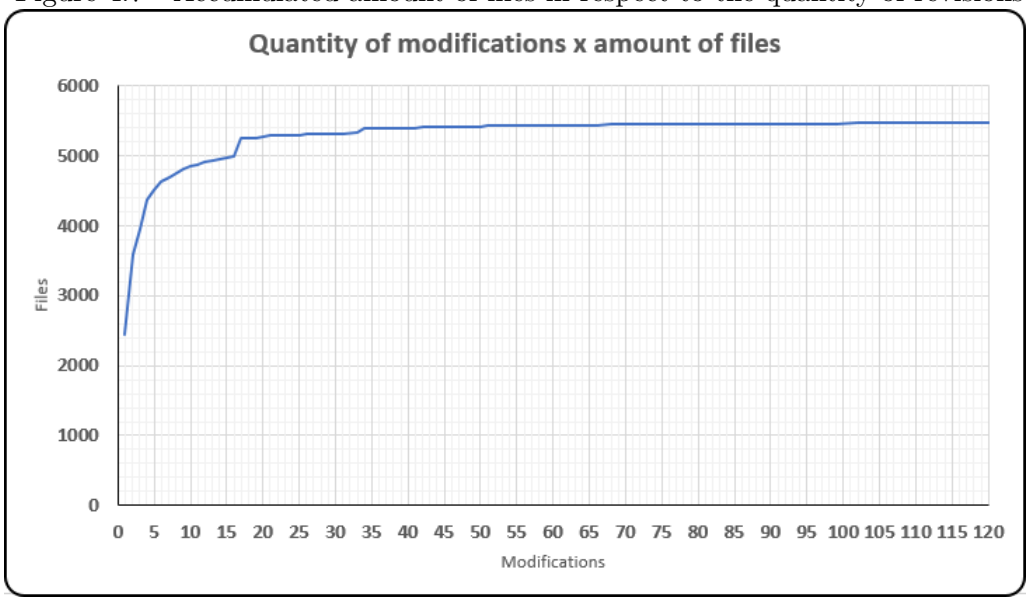


type experienced relocations inside schemas.

4.5.4 `XSDMiner2` and `SchemaCompare` data classification

Not all contracts in selected projects repositories have more than five revisions. In fact, a much greater amount of them do not satisfy this condition. This can be verified in Figure 4.7, where it is shown that around 4400 documents have four or less revisions. Considering that projects have a total of 5483 WSDL/XSD contracts, this is significative: a huge amount of contracts are not modified frequently.

Figure 4.7 - Accumulated amount of files in respect to the quantity of revisions



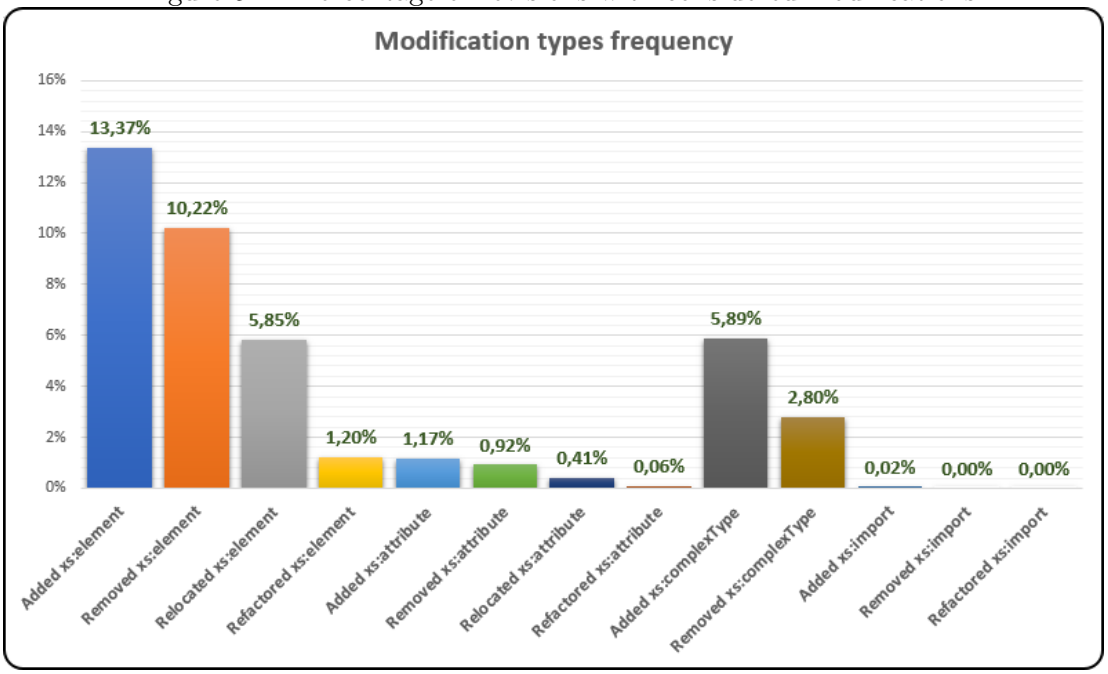
5 RESEARCH RESULTS

Data outputted from `XSDMiner2` and `SchemaCompare` were analyzed in order to answer the research questions listed in Section 4.1. So, each of the following subsections deals with one RQ.

5.1 RQ1 – What is the occurrence rate of each XSD modification type?

While some revisions change the metrics defined for this research, many of them do not. As so, the percentage of revisions with each modification type according to the total amount of revisions (for projects altogether) was calculated. To answer this RQ, these numbers were put together in the chart shown in Figure 5.1, obtained by querying the database table where `SchemaCompare` class stores its output.

Figure 5.1 - Percentage of revisions with considered modifications



It is made clear by chart in Figure 5.1 that `xs:element` tags experience much more modifications than the others. Moreover, the quantity of revisions with considered changes (in percentage), analyzed per modification types, do not exceed 14% of the total amount of files revisions. Tag `xs:complexType` do not experience refactoring (as defined for this research), as it does not have a property "name" (which is analyzed for refactoring), and `xs:import` tags are almost never changed.

Chart bars are not independent: different types of modifications often occur at the same commit. Even if they were, the sum of the percentages is not 100%, which indicates that there are other types modifications occurring with contracts and different from the ones analyzed in the present research. In fact, `SchemaCompare` class compared 10814 different commits from every project, and 6973 of them do not deal with any considered modification type. In relative numbers, 64.48% of commits are related to other changes not addressed here.

5.2 RQ2 – With which pace XSD modifications include or exclude information?

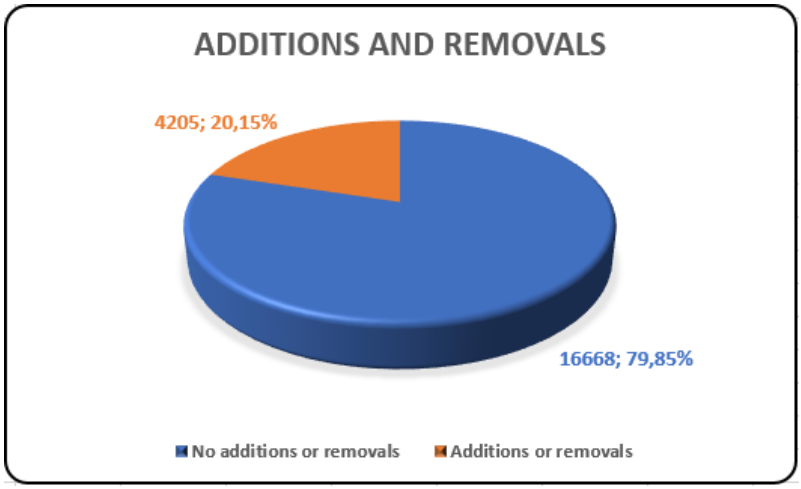
Contracts revisions bringing new information or lacking data when compared to its previous version are common. In terms of XML tags, it is related to the addition or removal of each tag type considered. When such modification occurs, a solution can be to include a "translation middleware" between the client and the service in order to make the necessary transformations among messages elements from different contract revisions. Even doing so, this might not fulfill the requirements of both nodes, imposing the need to adapt the involved systems to the new contract version. Thus, these changes have great relevance in web services communications.

Adjust messages to respect distinct contract revisions is possible using transformations. When a tag is removed from the contract (or not already present) and it is part of service request messages, conversion is easier: if the consumer sends requests for data not delivered anymore (or not already defined to be delivered), the middleware can filter them out, forward this adjusted request, receive the reply message and adjust it to deliver with default or null values for client expected informations. For added tags used in reply messages, it can just exclude the information from the response and deliver it without these new data. In case that requests and/or replies are just restructured into different new tags, the middleware can rearrange them and adapt to the new fields.

Even with all the workarounds mentioned, the transformations do not fulfill every use case of the service. An example of them is when new information is expected by the service and middleware can not infer from the request, or when data not delivered by the service is relevant for the client and should not be null or with a default value. Thus, adjustments in client and/or server nodes are mandatory and message conversions are not an alternative anymore. For many cases, it solves the problem and helps to maintain integration, but is not a sufficient solution.

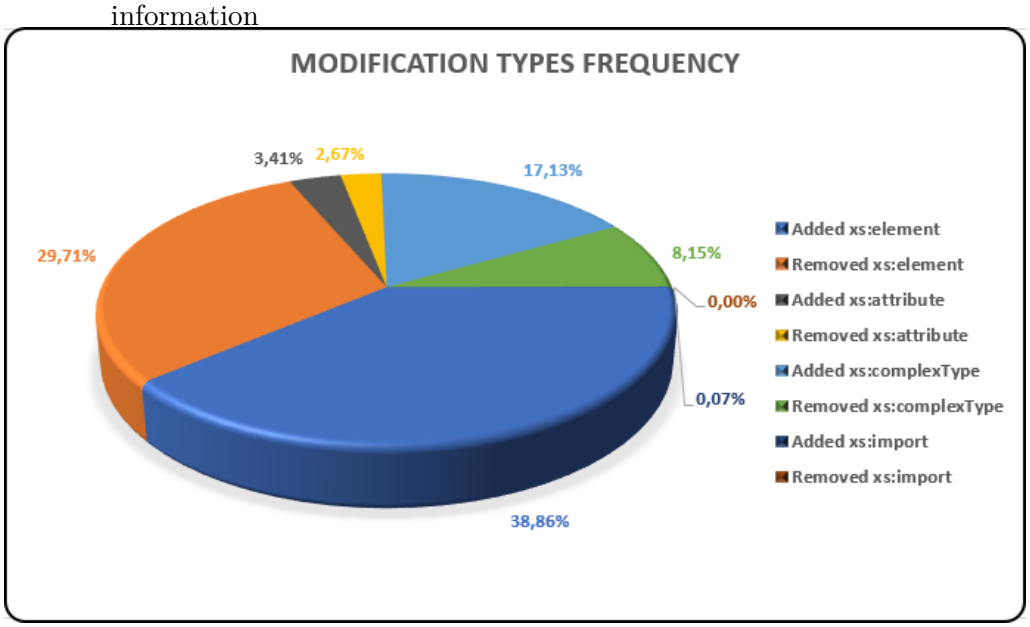
In RQ1 each modification was analyzed and it was found that many of them could occur at the same commit. To answer RQ2, though, it must be verified the presence or absence of the modification types that changes exchanged information. To do this, a query was performed to the database, in order to gather the amount of revisions with addition or removal of tags and the quantity of revisions where those changes did not happen. The result is presented in Figure 5.2, which shows that only around 20% of contracts revisions change documents contents.

Figure 5.2 - Contracts revisions distribution according to exchanged information changing



Regarding only the revisions with the modifications of interest to this RQ, it was calculated the percentage of each modification type, which is illustrated in Figure 5.3. It is made clear that modifications in `xs:element` tags represent the largest part of content changes found, and the `xs:import` tags the smallest part with almost no changes. It is also verified that for around 40% of those modifications it might be possible to apply message transformations to keep client-service integration (not considering consistency). It must be considered, though, the particular cases already mentioned, when data not delivered to client is important to it or when service expects information from the request and the middleware can not infer from the client message.

Figure 5.3 - Distribution of modification types that change WSDL/XSD exchanged information



5.3 RQ3 – How is the distribution of modifications among commits?

Changes in contracts do not respect a common pace between commits. In order to verify how is the distribution of modifications, it was calculated the mean value of quantity of changes per commit for all files, and analyzed the standard deviation to check concentration of changes. Data from this processing is presented in Appendix B and should be interpreted as: *files from "project" experience an average of "mean" modifications per commit, with this value varying to "standard deviation" limits.*

Checking data from Appendix B, it can be noticed that 11 of the 139 projects analyzed have no changes of the types considered in this research. Also, 121 of the 139 projects have standard deviation values greater than their mean values, which is an indicator that there is no consistency on the modifications frequency among different commits: while some have greater amounts of changes, others have fewer numbers, sometimes zero. Besides, 7 projects have standard deviations lower than their mean values but close to it, which can extend the inference of the 121 projects to them.

Modifications can happen in different files in the same commit, and the number of files that experience changes at each commit varies. The analysis of this behaviour was done by calculating the mean value of the amount of contracts modified at each commit, and the concentration of changes was verified by computing the standard

deviation of those numbers. Data from this processing is listed in Appendix C and should be interpreted as: *commits from "project" have modifications distributed in an average of "mean" files, with this quantity varying to "standard deviation" limits.*

Analyzing data in Appendix C shows that 11 projects have standard deviation values greater than the mean values. On the other hand, 127 of the other 128 projects (the exception is project `dssp-client`) have a low average of files being modified at each commit (the higher amount is less than 7). Thus, this indicates that modifications are usually well distributed among distinct contracts and not concentrated in few documents.

Conclusions obtained so far are extended by the graphs in Figure 5.4 and Figure 5.5. Disregarding the revisions with none of the considered modification types (already discussed in Section 5.3) and eventual outliers, it shows that 29.81% of commits have between 1 and 10 modifications, 3.84% between 11 and 25 modifications, and 0.59% more than 26 modifications. This leads to infer that a short number of commits have a high concentration of modifications, and that a great amount of revisions is related to a few number of changes.

Figure 5.4 - Distribution of modification types that change WSDL/XSD semantic

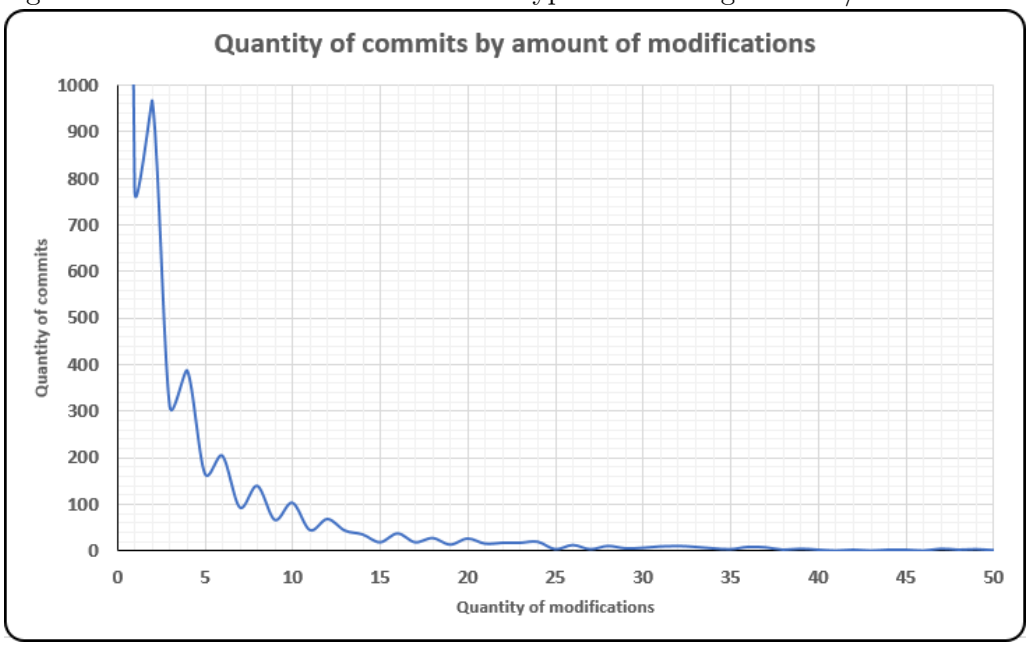
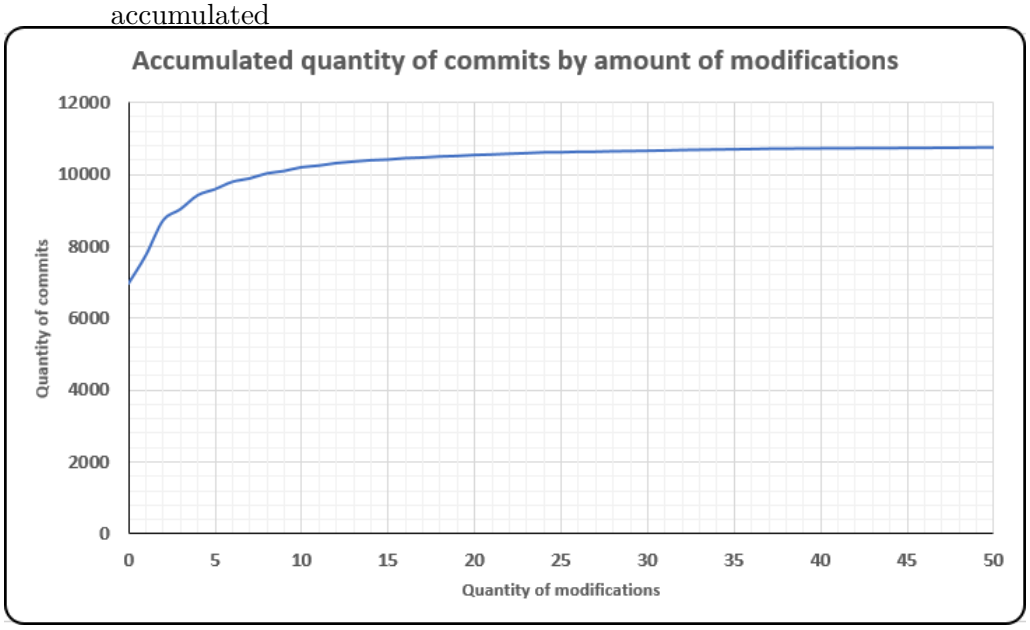


Figure 5.5 - Distribution of modification types that change WSDL/XSD semantic – accumulated



6 CONCLUSIONS

This dissertation aims to describe the behaviour of modifications in web services contracts schemas defined inside or outside WSDL documents and using XSD format, in order to provide a guidance to the development of web services to reduce adaptation effort after the natural evolution of contracts. To achieve this goal, a base of 139 GitHub projects were established to be analyzed, and code implementation was done to extract metrics and gather knowledge from the numbers obtained.

It was concluded that `xs:element` tags are the ones that experiences the most changes during the life cycle of the projects (followed by `xs:complexType` tags), and the numbers of those modifications are greater for additions and removals of tags. The other analyzed tags have less modifications, but they also have greater amounts of changes for additions and removals. This indicates that, in general, changes in tags tend to change the semantic of the contracts, by including new information or excluding data needed before.

From the numbers gathered in this research, it was detected that WSDL/XSD documents tend to experience a low number of modifications at each revision. This indicates that changes are not concentrated, and might lead to the interpretation that major revisions are not a tendency in web services projects. It might present a point to be addressed by developers, as it can compromise the client-provider integration by constantly compelling both nodes to readjust their systems to each contract revision.

Another conclusion was obtained from the amount of contracts modified at each commit. It was found that a huge amount of commits are related to a small number of documents. Since a lot of contracts are modified during the life cycle of projects, it indicates that commits are not always related to the same contracts, and revisions are not equally distributed among them. This fact might guide developers and project managers to restructure their development plannings, as it can force the nodes to continuously readjust different parts of their systems to avoid breaking the communication.

From all the results gathered, it can be inferred that web services projects, in general, do not have a well defined contract versioning schedule. This ultimately impacts on service nodes not being prepared to an upcoming change, demanding time to reestablish integration. Thus, the present research gives developers a new knowledge over their development profiles regarding contracts, which should be used

by them to adjust their development planning in order to avoid problems on service consumption.

Results showed that a large percentage of revisions are not related to semantic changing in contracts. This fact allows the usage of the architecture proposed by França and Guerra (2017) for those cases. The most frequent modification types can guide the evolution of a tool like Chrysalis (DUARTE, 2010) which, in conjunction with the mentioned architecture, uses, for now, one eXtensible Stylesheet Language for Transformation (XSLT) file per modification. As concluded from the present research, a large set of changes can occur in a single commit, which reveals that an optimization towards the reduction on the amount of the necessary XSLT files can be very important in a real scenario.

Not every aspect of changing in contracts were addressed in this research, which was verified by the numbers gathered with the proceeded analysis: only 25.52% of every existing modification was analyzed. This shows that more research can be done to evaluate other aspects of contracts evolution. Nevertheless, it is expected that the methodology defined and executed here serve as a guidance for further analysis, being extended with new techniques, metrics and classes.

6.1 Contributions

This research leaves some useful contributions, as follows:

- The methodology defined and applied can be reused for further analysis. Other types of XML elements and modifications can be verified with some adaptations in the implemented code, maintaining the walkthrough towards the necessary inferences.
- As already said, results obtained are indicative of web services projects not having a well defined contract versioning schedule. This dissertation brings new knowledge about this, which should be considered by web services developers when planning the evolution of the contracts of their projets.
- Analyzed XSD elements provided useful information regarding the behaviour of web services development. However, the elements and the types of modifications verified are 25% of every modification that schemas experience. This research, then, suggests that more analysis can be done over the evolution of web services contracts and indicates that more new knowledge about it can still be discovered.

- All code implemented here, along with the metrics database, is open-source and freely available at GitHub. They can be reused and extended for future related researches.

- The obtained results can be taken into account by web service developers when planning the system development, to improve integration and reduce communication breaks when evolving their contracts.

6.2 Future work

As said before, there is still a wide range of modifications in contracts to be analyzed in order to provide more knowledge to web service development improvement. So, some research suggestions are given here.

In the present study, four XML elements present in WSDL contracts were analyzed, and this analysis was done over the amounts of changes. Many other modifications, though, are not only related to those types of changing: XSD schemas can contain, for example, restrictions to `xs:element` tags, which can be inside the node or referenced as a `xs:simpleType` external tag. This and other examples of tags remain not analyzed. Future research should consider them and verify if they have more undisclosed useful knowledge.

Refactoring, in this study, was considered as being a change in a tag "name" attribute. However, it is a much wider theory, which basically involves modification of document definitions and structure without changing its conception. For example, changing the name of a tag and keeping its definitions is considered a refactoring. Another type of refactoring is when a restriction is applied to a `xs:element` tag: the restriction can be defined inside it or outside (being correctly referenced), and a modification from one type of definition to another is also considered a refactoring. Not all variants of refactoring were considered here. Even renaming used a similarity algorithm to compare names and did not verified the other tag characteristics to consider a pair to match in two consecutive revisions. Nevertheless, it already found relevant knowledge. It is expected, then, that a research considering every possibility of refactoring should bring some more useful information.

The comparison between consecutive revisions used a Java class that tries to match nodes in two documents, showing the differences found in the process. The implemented code that uses it has an extra task to find nodes that were just moved inside schema (which was called here a relocation). It is desirable that

comparisons are always made using equivalent nodes, to guarantee that no wrong matchings occurs. This can be achieved by implementing Java code that creates the XSD structure as a tree and compares two documents using it, which might help addressing the refactoring problem.

Analysis were done over WSDL and XSD documents, regarding only the data types used to exchange messages in a SOA environment. However, WSDL describes other aspects of the service, like the connection endpoints and the operations provided by the service. These were not studied. Research over them should be undertaken to find out the behaviour of every aspect of the service.

Web services projects were selected over the GitHub platform, as MetricMiner only operates over this type of software repository. However, other repository platforms are widely used, like Subversion. The amount of projects studied here is large, but could be greater if other repository types were used. An extension of this research should be considered, in order to find out if projects in other repository platforms follow the same behaviour.

REFERENCES

ABATE, P.; COSMO, R. D.; GESBERT, L.; Le Fessant, F.; TREINEN, R.; ZACCHIROLI, S. Mining component repositories for installability issues. In: **Proceedings of the 12th Working Conference on Mining Software Repositories**. Florence, Italy: IEEE, 2015. p. 24–33. Available from: <http://www.dicosmo.org/preprints/msr-2015-distcheck.pdf>. 20

ALMEIDA, D. B. F. C.; GUERRA, E. M. Evolution of XSD documents and their variability during project life cycle: a preliminary study. In: _____. **Computational Science and Its Applications – ICCSA 2016: 16th International Conference, Beijing, China, July 4-7, 2016, Proceedings, Part IV**. Springer International Publishing, 2016. p. 392–406. ISBN 978-3-319-42089-9. Available from: <http://dx.doi.org/10.1007/978-3-319-42089-9_28>. 23

ANICHE, M. F.; SOKOL, F. Z.; GEROSA, M. MetricMiner: supporting researchers in mining software repositories. **IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)**, p. 142–146, 2013. 3, 20, 24

BARBOSA, H. A.; SILVA, L. R. M. A step beyond visualization: how to ingest Meteosat second generation satellite data and products into McIDAS-V, ILWIS and TerraMA2. **Journal of Hyperspectral Remote Sensing**, v. 4, n. 1, p. 01–18, 2014. 7

BENDINI, H. N.; MORAES, W. S.; COSTA, S.; LOPES, E. S.; KÖRTING, T. S.; FONSECA, L. M. G. Proposta de sistema de monitoramento da Sigatoka-Negra baseado em variáveis ambientais utilizando o TerraMA2. In: DAVIS JR., C. A.; FERREIRA, K. R. (Ed.). **Proceedings of the XV Brazilian Symposium on GeoInformatics**. Campos do Jordão, Brazil: MCTI/INPE, 2014. p. 168–173. ISSN 2179-4820. 7

BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C. M.; MALER, E.; YERGEAU, F. Extensible Markup Language (XML). **World Wide Web Consortium Recommendation REC-xml-19980210**, v. 16, 1998. 2nd ed. Available from: <http://www.w3.org/TR/1998/REC-xml-19980210>. 16

CÂMARA, G.; VINHAS, L.; FERREIRA, K. R.; QUEIROZ, G. R. D.; SOUZA, R. C. M. D.; MONTEIRO, A. M. V.; CARVALHO, M. T. D.; CASANOVA,

M. A.; FREITAS, U. M. D. TerraLib: an open source GIS library for large-scale environmental and socio-economic applications. In: HALL, G. B.; LEAHY, M. G. (Ed.). **Open source approaches in spatial data handling**. Berlin, Germany: Springer, 2008. p. 247–270. 6

CARVALHO, L. P. S.; NOVAIS, R.; NETO, M. G. de M. VisMinerService: A REST web service for source mining. In: **III Workshop de Visualização, Evolução e Manutenção de Software (VEM)**. Belo Horizonte, Brazil: [s.n.], 2015. p. 89–96. Available from: <http://vem2015.dcc.ufla.br/wp-content/uploads/2015/08/2015_vem_carvalho_et_al.pdf>. 20

CHRISTEN, M. **Conhecendo melhor as capacidades do Enterprise Service Bus**. Microsoft Corporation, 2009. Available from: <https://msdn.microsoft.com/pt-br/library/dd920288.aspx>. 14

COMITÊ DE PLANEJAMENTO DA INFRAESTRUTURA NACIONAL DE DADOS ESPACIAIS. **Plano de Ação para Implantação da Infraestrutura Nacional de Dados Espaciais**. Rio de Janeiro, Brazil: Comissão Nacional de Cartografia (CONCAR), 2010. Available from: <http://www.concar.gov.br/pdf/PlanoDeAcaoINDE.pdf>. 5

CONNER, P.; ROBINSON, S. **Service-oriented architecture**. Google Patents, 2007. US Patent App. 11/388,624. Available from: <https://www.google.com/patents/US20070011126>. 11

CREPANI, E.; MEDEIROS, J. Imagens CBERS+, imagens SRTM+, mosaicos GeoCover e LANDSAT em ambiente SPRING e TerraView: sensoriamento remoto e geoprocessamento gratuitos aplicados ao desenvolvimento sustentável. In: **XII Simpósio Brasileiro de Sensoriamento Remoto**. Goiânia, Brazil: INPE, 2005. v. 12. 7

DIRETORIA DE SERVIÇO GEOGRÁFICO DO EXÉRCITO. **Especificação Técnica para Estruturação de Dados Geoespaciais Vetoriais (ET-EDGV)**. Rio de Janeiro, Brazil: Comssão Nacional de Cartografia (CONCAR), 2010. 6

DUARTE, A. P. **Ferramenta para refatoração de documentos XML**. Monograph (Graduation in Computer Engineering) — Instituto Tecnológico de Aeronáutica, São José dos Campos, São Paulo, Brazil, 2010. 56

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. 180 p. Thesis (PhD in Information and Computer

Science) — University of California – Irvine, Irvine, California, USA, 2000. Available from: <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>. 15

FRANÇA, D. S.; ANICHE, M.; GUERRA, E. M. Como o formato de arquivos XML evolui? um estudo sobre sua relaço com o código-fonte. **3rd Workshop on Software Visualization, Evolution, and Maintenance (VEM)**, Belo Horizonte-MG, Brazil, 2015. 2, 4, 7, 8

FRANÇA, D. S.; GUERRA, E. M. **Modelo arquitetural para gerenciamento de versões de contratos de serviços web**. 79 p. Dissertation (M. Sc. in Applied Computing) — Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, São Paulo, Brazil, 2017. 56

FRANK, D.; LAM, L.; FONG, L.; FANG, R.; KHANGAONKAR, M. Using an interface proxy to host versioned web services. In: **Proceedings of the 2008 IEEE International Conference on Services Computing**. Honolulu, USA: IEEE, 2008. v. 2, p. 325–332. 4

GALA-PÉREZ, S.; ROBLES, G.; GONZÁLEZ-BARAHONA, J. M.; HERRAIZ, I. Intensive metrics for the study of the evolution of open source projects: case studies from Apache Software Foundation projects. In: **Proceedings of the 10th Working Conference on Mining Software Repositories**. San Francisco, USA: [s.n.], 2013. Available from: <http://oa.upm.es/14698/>. 20

GLASS, R. L. The state of the practice of software engineering. **IEEE Software**, IEEE Computer Society, Washington, USA, v. 20, n. 6, p. 20 – 21, 2003. 19

HASSAN, A. E. **Mining software repositories to assist developers and support managers**. 300 p. Thesis (PhD in Computer Science) — University of Waterloo, Waterloo, Canada, 2004. Available from: <https://uwspace.uwaterloo.ca/bitstream/handle/10012/1017/aeehassa2004.pdf?sequence=1>. 19

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO 8879:1986: Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)**. 1986. 16

KAGDI, H.; COLLARD, M. L.; MALETIC, J. I. 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**,

John Wiley & Sons, Ltd., v. 19, n. 2, p. 77–131, 2007. ISSN 1532-0618. Available from: <http://dx.doi.org/10.1002/smr.344>. 19

LEITNER, P.; MICHLMAYR, A.; ROSENBERG, F.; DUSTDAR, S. End-to-end versioning support for web services. In: **Proceedings of the 2008 IEEE International Conference on Services Computing**. Honolulu, USA: IEEE, 2008. v. 1, p. 59–66. 4

LIMA, F. A.; ALMEIDA, L.; BRAGA, F. L.; NERY, C. V. M. Utilização do sistema de informações geográficas Terraview para delimitação da bacia hidrográfica do Rio Vieira, Montes Claros–MG. **Simpósio Regional de Geoprocessamento e Sensoriamento Remoto – Geonordeste, VI**, 2012. 7

MENGE, F. Enterprise Service Bus. In: **Proceedings of the 2007 Free and Open Source Software Conference**. Sankt Augustin, Germany: FrOSCon, 2007. v. 2, p. 1–6. Available from: <https://programm.froscon.org/2007/attachments/15-falko_menge_-_enterpise_service_bus.pdf>. 14

OLATUNJI, S. O.; IDREES, S. U.; AL-GHAMDI, Y. S.; AL-GHAMDI, J. S. A. Mining software repositories – a comparative analysis. **International Journal of Computer Science and Network Security**, IJCSNS, Seoul, Korea, v. 10, p. 161 – 174, 2010. N. 8. Available from: <http://paper.ijcsns.org/07_book/201008/20100826.pdf>. 19

OPEN GEOSPATIAL CONSORTIUM. **OGC standards and supporting documents**. 2006. Available from: <http://www.opengeospatial.org/standards>. 5

PAPAZOGLOU, M. P. The challenges of service evolution. In: BELLAHSÈNE, Z.; LÉONARD, M. (Ed.). **Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE)**. Berlin, Germany: Springer, 2008. p. 1–15. ISBN 978-3-540-69534-9. Available from: <http://dx.doi.org/10.1007/978-3-540-69534-9_1>. 8

PATIG, S. Design of SOA services: experiences from industry. In: CORDEIRO, J.; RANCHORDAS, A.; SHISHKOV, B. (Ed.). **Software and Data Technologies**. Springer, 2011, (Communications in Computer and Information Science, v. 50). p. 150–163. ISBN 978-3-642-20115-8. Available from: <http://dx.doi.org/10.1007/978-3-642-20116-5_12>. 11

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. RESTful web services vs. "big" web services: making the right architectural decision. In: **Proceedings of the**

17th International Conference on World Wide Web. New York, USA: ACM, 2008. (WWW '08), p. 805–814. ISBN 978-1-60558-085-2. Available from: <http://doi.acm.org/10.1145/1367497.1367606>. 12, 15, 18

PETERS, R.; ZAIDMAN, A. Evaluating the lifespan of code smells using software repository mining. In: **Proceedings of the 16th European Conference on Software Maintenance and Reengineering**. Szeged, Hungary: IEEE Computer Society, 2012. p. 411–416. ISSN 1534-5351. 20

POUTSMA, A.; EVANS, R.; RABBO, T. A. Why contract first? In: **Spring Web Services Reference Documentation**. San Francisco, USA: SpringSource, 2007. Available from: <http://docs.spring.io/spring-ws/site/reference/html/why-contract-first.html>. 18

QIU, D.; LI, B.; SU, Z. An empirical analysis of the co-evolution of schema and code in database applications. In: **Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering**. Saint Petersburg, Russia: Association for Computer Machinery (ACM), 2013. p. 125–135. 8

RAY, B.; NAGAPPAN, M.; BIRD, C.; NAGAPPAN, N.; ZIMMERMANN, T. The uniqueness of changes: characteristics and applications. In: **Proceedings of the 12th Working Conference on Mining Software Repositories (MSR)**. Florence, Italy: IEEE, 2014. Available from: <http://research.microsoft.com/apps/pubs/default.aspx?id=232407>. 20

ROMANO, D.; PINZGER, M. Analyzing the evolution of web services using fine-grained changes. In: **Proceedings of the 19th International Conference on Web Services**. IEEE, 2012. p. 392–399. Available from: <https://ai2-s2-pdfs.s3.amazonaws.com/014a/89278437ab0fda58c717ed4ce14a7bddfd11.pdf>. 8

ROSIM, S.; LIMA, S. F. S.; MORAES, E. C.; JARDIM, A. C.; OLIVEIRA, J. R. de F. Aplicação do TerraHidro para delimitação automática de drenagem e limite das sub-bacias do rio Miranda. **Anais do 4º Simpósio de Geotecnologias no Pantanal, Bonito, MS. Embrapa Informática Agropecuária/INPE**, p. 405–412, 2012. 7

ROSIM, S.; OLIVEIRA, J. R. de F.; JARDIM, A. C.; CUELLAR, M. Z. Extração da drenagem da Região Nordeste utilizando o sistema TerraHidro. **Simpósio de Recursos Hídricos do Nordeste**, v. 12, 2014. 7

ROTEM-GAL-OZ, A. **Bridging the impedance mismatch between business intelligence and service-oriented architecture**. Microsoft Developer Network, 2007. Available from: <https://msdn.microsoft.com/en-us//library/bb419307.aspx#XSLTsection128121120120>. 12

SAMPAIO, C. **SOA e WebServices em Java**. Rio de Janeiro, Brazil: Brasport, 2006. 1

SILVA, F. M.; GOMES, C.; CUELLAR, M. Z.; ALMEIDA, S. A. S.; AMORIM, R. F.; CARVALHO, M. J. M. Comportamento espacial do Índice de Desenvolvimento Humano no Rio Grande do Norte com uso do programa TerraView (desenvolvido pelo INPE). In: **XIII Simpósio Brasileiro de Sensoriamento Remoto**. Florianópolis, Brazil: INPE, 2007. 7

SPACCO, J.; STRECKER, J.; HOVEMEYER, D.; PUGH, W. Software repository mining with Marmoset: an automated programming project snapshot and testing system. In: **Proceedings of the 2005 International Workshop on Mining Software Repositories**. New York, USA: ACM, 2005. p. 1–5. ISBN 1-59593-123-6. Available from: <http://doi.acm.org/10.1145/1082983.1083149>. 20

WEERAWARANA., S.; CHRISTENSEN, E.; CURBERA, F.; MEREDI, G. **Web Services Description Language (WSDL) 1.1**. Cambridge, USA, 2001. Note 15, 2001. Available from: <http://www.w3.org/TR/wsdl>. 1, 13, 17

WORLD WIDE WEB CONSORTIUM (W3C). **Web Services Glossary**: W3C Working Group Note (February 11th, 2004). Cambridge, USA, 2004. Available from: <http://www.w3.org/TR/ws-gloss/>. 1

_____. **XML Schema**. Cambridge, USA, 2004. Available from: <http://www.w3.org/XML/Schema/>. 1, 16

_____. **SOAP Version 1.2 Part 1**: Messaging framework. Cambridge, USA, 2007. 2nd ed. Available from: <http://www.w3.org/TR/soap12>. 13

YING, A.; MURPHY, G.; NG, R.; CHU-CARROLL, M. Predicting source code changes by mining change history. **IEEE Transactions on Software Engineering**, v. 30, n. 9, p. 574–586, 2004. ISSN 0098-5589. Available from: <http://trese.cs.utwente.nl/publications/files/04692004-tse-mine-change.pdf>. 19

APPENDIX A – TOTAL CONTRACTS REVISIONS WITH EACH TYPE OF MODIFICATION PER PROJECT

Caption: (A) added `xs:element`; (B) removed `xs:element`; (C) relocated `xs:element`; (D) refactored `xs:element`; (E) added `xs:attribute`; (F) removed `xs:attribute`; (G) relocated `xs:attribute`; (H) refactored `xs:attribute`; (I) added `xs:complexType`; (J) removed `xs:complexType`; (K) added `xs:import`.

Project	A	B	C	D	E	F	G	H	I	J	K
acplugins	24	14	11	3	0	0	0	0	9	2	0
alloc	0	0	0	0	0	0	0	0	0	0	0
amadeus-ws-client	0	0	0	0	0	0	0	0	0	0	0
animated-batman	0	2	2	0	0	0	0	0	2	0	0
apache-ode	12	4	4	2	8	0	0	0	5	1	0
apollo	203	215	146	47	1	0	0	0	178	128	5
battleship	7	2	0	0	0	0	0	0	1	0	0
bearded-archer	9	7	3	0	0	0	0	0	1	1	0
betsy	3	1	1	0	0	0	0	0	0	0	0
bfcp	2	2	1	0	0	0	0	0	0	0	0
BPEL-splitting	3	1	0	1	0	0	0	0	0	0	0
bpmlabs-services	3	4	3	0	0	0	0	0	1	2	0
btc4j-ws	0	1	0	0	0	0	0	0	2	1	0
bypass-stored-value	0	0	0	0	0	0	0	0	0	0	0
c3pr	80	92	58	7	6	6	1	0	41	30	0
caaers	211	205	147	16	14	4	1	0	100	44	0
caarray	28	28	5	8	14	12	2	2	15	13	0
cagrid	380	227	105	29	69	59	20	5	178	50	0
cagrid-core	37	33	10	0	2	5	2	0	12	1	0
cagrid-grid-incubation	51	26	10	2	4	0	0	0	19	1	0
CalendarPortlet	0	0	0	0	0	0	0	0	0	0	0
CallCenter	17	5	5	2	0	0	0	0	7	0	0

Project	A	B	C	D	E	F	G	H	I	J	K
camel	21	9	2	1	20	16	10	2	7	2	0
capture-hpc	0	1	1	0	0	0	0	0	7	5	0
carbon-analytics-common	35	32	19	5	0	0	0	0	10	7	0
carbon-apimgt	17	14	9	7	0	0	0	0	1	1	0
carbon-business-messaging	5	2	1	1	0	0	0	0	4	3	0
carbon-business-process	30	17	7	0	2	0	0	0	6	2	0
carbon-data	4	1	0	0	0	0	0	0	0	0	0
carbon-deployment	11	14	12	0	0	0	0	0	4	1	0
carbon-event-processing	26	25	20	11	0	0	0	0	4	4	0
carbon-identity	74	71	47	7	0	0	0	0	20	13	0
carbon-identity-framework	3	0	0	0	0	0	0	0	0	0	0
carbon-metrics	6	8	5	0	0	0	0	0	1	1	0
carbon-storage-management	7	8	6	1	0	0	0	0	5	4	0
cartoweb3	24	17	6	2	0	0	0	0	9	4	0
centreon-engine	7	13	11	2	0	0	0	0	15	5	0
cerebrum	0	0	0	0	0	0	0	0	0	0	0
channelAdvisorAccess	19	17	11	6	0	0	0	0	10	9	0
CONNECT-Webservices	0	0	0	0	0	0	0	0	0	0	0
Consent2Share	0	0	0	0	0	0	0	0	0	0	0
corral	18	21	10	0	0	0	0	0	8	3	0
csla	2	4	2	0	1	0	0	0	4	2	0
cws-esolutions	3	3	0	0	0	0	0	0	3	3	0
CxTest	5	0	0	0	0	0	0	0	4	0	0
dblog	29	8	3	1	0	0	0	0	20	1	0
DemoGames	9	6	1	0	0	0	0	0	2	1	0
devstudio-tooling-bps	0	0	0	0	0	0	0	0	0	0	0
DistMM	2	1	0	0	0	0	0	1	1	1	0

Project	A	B	C	D	E	F	G	H	I	J	K
DL-Learner	1	0	0	0	0	0	0	0	1	0	0
doms-bitstorage	12	10	4	3	1	1	1	0	2	3	0
doms-server	33	21	16	1	1	0	0	0	18	4	0
droolsjbpm-integration	32	13	2	0	20	13	10	0	10	2	0
dssp-client	0	3	2	0	0	0	0	0	0	1	0
ebay-api-sdk-php	0	0	0	0	0	0	0	0	0	0	0
ebmsadapter	0	0	0	0	0	0	0	0	0	0	0
EDLProvider	4	5	4	3	0	1	1	0	2	1	0
EECloud	8	5	4	1	0	0	0	0	0	0	0
eqtl	0	0	0	0	0	0	0	0	0	0	0
eucalyptus	62	35	20	3	0	0	0	0	24	12	0
eve-intel-map	6	6	3	1	0	0	0	0	1	1	0
ezags-xsd	8	10	7	1	0	0	0	0	2	2	0
gds-mis	32	48	32	3	0	0	0	0	6	6	0
Gemma	2	0	0	0	0	0	0	0	0	0	0
GNDMS	46	44	30	1	4	10	2	0	20	18	0
gsn	11	9	1	0	0	4	1	0	5	2	0
Habitat	5	1	1	0	0	0	0	0	2	0	0
HandHeldInventory	4	3	0	0	0	0	0	0	0	0	0
helium	2	0	0	0	0	0	0	0	1	0	0
hotell	7	3	2	0	0	0	0	0	1	1	0
htcondor	32	32	17	0	5	3	1	0	7	2	0
idecore	14	15	8	3	0	0	0	0	12	8	0
identity-inbound-auth-oauth	3	0	0	0	0	0	0	0	0	0	0
ihub	9	7	5	0	0	0	0	0	6	0	0
incubator-stratos	10	7	3	0	4	4	0	0	5	1	0
IQ-Champions	8	3	2	0	0	0	0	0	2	0	0

Project	A	B	C	D	E	F	G	H	I	J	K
iws	31	24	17	1	0	0	0	0	46	36	0
jdk7u-jdk	0	0	0	0	0	0	0	0	0	0	0
JoinToPlayClient	6	5	2	0	0	0	0	0	2	1	0
juddi	0	0	0	0	0	0	0	0	0	0	0
kask-kiosk	30	18	12	4	0	0	0	0	13	1	0
kc	22	10	7	1	2	0	0	0	6	2	0
kc.preclean	22	10	7	1	2	0	0	0	6	2	0
kdepim-ktimetracker-akonadi	13	21	20	4	7	3	2	0	7	3	0
kdepim-noakonadi	13	21	20	4	7	3	2	0	7	3	0
kopete	0	0	0	0	0	0	0	0	0	0	0
Liltarp-Assignment	10	10	8	1	0	0	0	0	3	2	0
lime-security-powerauth	30	8	3	1	0	0	0	0	0	0	0
lolsoap	5	8	6	0	4	3	0	0	4	2	0
magento2	38	43	23	4	19	23	16	0	33	18	0
mcgssg1_servidor	5	3	2	0	0	0	0	0	5	0	0
MDW	17	18	4	0	0	0	0	0	1	2	0
MDW_BattleShips	2	1	0	2	0	0	0	0	0	0	0
MOOLGOSS	9	5	2	1	0	0	0	0	2	0	0
mplus-api-wsdl	12	13	11	2	0	0	0	0	9	4	0
mule-cookbook	12	16	8	0	0	0	0	0	10	8	0
mule-wss-soap-example	0	0	0	0	0	0	0	0	0	0	0
multidatabase	9	4	1	0	0	0	0	0	0	0	0
myPress	10	7	5	0	0	0	0	0	2	0	0
NE-HSCIE-Core	13	5	2	0	0	0	0	0	7	1	0
NOTES-WEB	29	4	1	2	0	0	0	0	10	0	0
OCHP	26	18	9	1	3	0	0	0	10	2	0
ode	23	13	7	2	3	1	0	0	10	4	0

Project	A	B	C	D	E	F	G	H	I	J	K
opencover	5	5	1	2	0	0	0	0	2	1	0
OpenNos	3	2	0	0	0	0	0	0	0	0	0
openspecimen	1	6	2	0	2	4	3	0	0	3	0
otrs-gitimport-test	7	9	4	2	0	0	0	0	5	2	0
oxalis	0	0	0	0	0	0	0	0	0	0	0
petals-se-activiti	13	4	0	0	0	1	1	0	2	0	0
petnet-web	62	21	2	1	0	0	0	0	10	3	0
php-amadeus	2	10	8	0	0	0	0	0	8	0	0
PizzaWaiter	22	12	9	0	0	0	0	0	9	1	0
polymony	11	5	0	1	0	0	0	0	2	0	0
processmaker	18	2	0	0	0	0	0	0	4	0	0
ProcessManager	4	1	0	0	8	5	4	1	4	1	0
product-as	1	1	0	0	0	0	0	0	0	0	0
pwm	0	1	1	0	0	0	0	0	0	0	0
python-lemonway	4	0	0	0	0	0	0	0	0	0	0
rconomic	2	0	0	1	0	0	0	0	2	0	0
rig-client	5	5	2	1	0	0	0	0	2	1	0
RSB	6	2	2	0	4	2	1	0	5	0	0
s2	5	4	3	0	0	0	0	0	2	2	0
scape	9	8	6	0	2	3	0	1	6	2	0
scheduling-server	40	36	26	2	1	2	0	0	14	9	0
sci-flex-synapse-esper-plugin	4	3	1	0	0	0	0	0	3	2	0
sdk-client-tools-protex	8	2	0	0	0	0	0	0	6	0	0
SEServerExtender	10	10	8	0	0	0	0	0	2	1	0
SevenUpdate	11	10	4	3	1	0	0	0	1	0	0
simias	102	41	17	7	0	0	0	0	17	9	0
sipXtapi	19	17	10	2	0	0	0	0	14	11	0

Project	A	B	C	D	E	F	G	H	I	J	K
sones	8	10	4	5	3	3	3	0	2	3	0
sponsored-search-api-documents	0	1	1	0	0	0	0	0	0	0	0
staff	1	2	2	0	0	0	0	0	4	4	0
steve	1	2	1	1	0	0	0	0	0	0	0
stratos	35	40	17	0	0	0	0	0	6	1	0
sts	0	0	0	0	0	0	0	0	0	0	0
SwingDriver	3	1	0	0	0	0	0	0	2	1	0
Sync	8	8	5	0	0	0	0	0	1	1	0
sysmgrt	6	3	0	0	0	0	0	0	4	4	0
taverna-grid	20	24	13	2	0	0	0	0	7	6	0
tempo	30	18	8	0	0	0	0	0	5	1	0
Tennis-Middleware	8	3	2	2	0	0	0	0	2	1	0
Thesis	7	5	4	0	0	0	0	0	2	0	0
tracee	1	0	0	0	0	0	0	0	1	0	0
UltraMarket-System	5	5	4	0	0	0	0	0	5	2	0
UMProVolleyWeb	10	8	4	1	0	0	0	0	6	1	0
weblabdeusto	12	3	1	0	0	0	0	0	2	0	0
Websitepanel	1	0	0	0	1	1	1	0	0	0	0
wizard	23	13	13	4	0	0	0	0	3	0	0
wsdlgo	2	1	0	0	0	0	0	0	0	0	0
x-road-adapter-example	0	0	0	0	0	0	0	0	0	0	0
xibocms	0	0	0	0	0	0	0	0	0	0	0
xroad-catalog	1	4	2	0	0	0	0	0	2	3	0
ydn-api-documents	2	0	0	0	0	0	0	0	1	0	0
zato	6	9	6	1	0	0	0	0	0	0	0
zend-soap	0	0	0	0	0	0	0	0	0	0	0
Zero-K-Infrastructure	45	18	3	1	0	0	0	0	17	6	0

Project	A	B	C	D	E	F	G	H	I	J	K
zf2	0	0	0	0	0	0	0	0	0	0	0
zuora	17	13	0	0	0	0	0	0	2	2	0

APPENDIX B – STATISTICS FOR MODIFICATION QUANTITIES PER CONTRACTS AMONG COMMITS

Project	Mean	Standard deviation
acplugins	6.17	12.68
alloc	0.00	0.00
animated-batman	0.47	1.37
apache-ode	0.53	1.11
apollo	0.35	4.03
battleship	1.42	1.68
bearded-archer	2.57	3.36
betsy	0.38	0.72
BPEL-splitting	0.35	0.61
bpmlabs-services	3.63	4.66
btc4j-ws	1.00	1.67
bypass-stored-value	0.00	0.00
c3pr	4.88	8.92
caaers	4.94	10.30
caarray	1.48	3.24
cagrid	2.17	3.57
cagrid-core	2.36	3.52
cagrid-grid-incubation	2.22	3.07
CalendarPortlet	0.00	0.00
CallCenter	44.27	102.93
camel	1.41	1.99
capture-hpc	2.67	3.62
carbon-analytics-common	2.98	5.16
carbon-apimgt	0.78	1.30
carbon-business-messaging	1.00	1.05
carbon-business-process	1.22	1.44

Project	Mean	Standard deviation
carbon-data	1.22	1.99
carbon-deployment	3.19	4.63
carbon-event-processing	2.76	4.38
carbon-identity	2.28	6.20
carbon-identity-framework	0.56	0.88
carbon-metrics	2.33	3.98
carbon-storage-management	2.53	3.06
cartoweb3	3.33	5.34
centreon-engine	2.14	3.09
channelAdvisorAccess	11.20	19.12
corral	2.55	3.56
csla	3.30	9.49
cws-esolutions	3.22	8.45
CxTest	10.67	10.80
dblog	13.77	13.47
DemoGames	4.75	7.01
DistMM	1.57	2.44
DL-Learner	0.09	0.43
doms-bitstorage	1.49	3.22
doms-server	2.92	5.47
droolsjbpm-integration	2.00	2.87
dssp-client	2.75	4.53
ebmsadapter	0.00	0.00
EDLProvider	2.12	2.83
EECloud	1.26	3.07
eucalyptus	5.70	8.99
eve-intel-map	2.53	5.00

Project	Mean	Standard deviation
ezags-xsd	2.03	3.83
gds-mis	4.56	8.25
GNDMS	2.23	3.89
gsn	2.37	2.86
Habitat	4.83	2.86
HandHeldInventory	2.50	2.43
helium	1.80	3.49
hotell	2.21	3.12
htcondor	2.36	3.86
idecore	55.00	91.95
identity-inbound-auth-oauth	1.40	1.67
ihub	2.18	4.68
incubator-stratos	0.63	1.57
IQ-Champions	3.29	4.63
iws	7.58	16.20
jdk7u-jdk	0.00	0.00
JoinToPlayClient	7.40	10.49
juddi	0.00	0.00
kask-kiosk	12.03	25.58
kc	0.42	2.32
kc.preclean	0.42	2.32
kdepim-ktimetracker-akonadi	8.82	18.49
kdepim-noakonadi	8.82	18.49
kopete	0.00	0.00
Liltarp-Assignment	5.13	6.27
lime-security-powerauth	5.72	12.86
lolsoap	1.36	1.93

Project	Mean	Standard deviation
magento2	2.60	3.27
mcgssg1_servidor	1.00	1.25
MDW	2.48	3.76
MOOLGOSS	1.91	2.76
mplus-api-wsdl	37.00	40.48
mule-cookbook	4.93	5.07
mule-wss-soap-example	0.00	0.00
multidatabase	2.90	5.52
myPress	3.42	5.01
NE-HSCIE-Core	3.71	4.37
NOTES-WEB	17.69	40.39
OCHP	2.37	3.38
ode	0.55	1.70
opencover	2.50	3.97
openspecimen	0.80	2.20
otrs-gitimport-test	6.29	9.86
oxalis	0.00	0.00
petals-se-activiti	1.21	3.61
petnet-web	5.41	11.46
php-amadeus	1.04	4.87
PizzaWaiter	4.75	6.09
polymony	3.38	4.77
processmaker	2.00	1.22
ProcessManager	2.59	2.29
python-lemonway	8.50	10.66
rig-client	2.46	4.07
RSB	0.79	1.27

Project	Mean	Standard deviation
s2	21.20	27.13
scape	3.25	3.57
scheduling-server	6.58	13.56
sci-flex-synapse-esper-plugin	1.33	1.37
sdk-client-tools-protex	0.75	1.94
SEServerExtender	2.60	5.15
SevenUpdate	0.96	3.45
simias	4.00	6.14
sipXtapi	4.38	4.48
sones	7.13	7.82
staff	2.10	3.73
steve	1.00	2.13
stratos	2.56	3.83
sts	0.00	0.00
SwingDriver	4.75	10.81
Sync	1.45	4.63
sysmgrt	4.19	6.12
taverna-grid	2.00	4.21
tempo	2.19	2.64
Tennis-Middleware	6.00	8.77
Thesis	3.61	6.34
UltraMarket-System	3.55	2.70
UMProVolleyWeb	33.00	78.21
weblabdeusto	1.16	1.07
Websitepanel	0.80	2.53
wizard	6.31	12.11
wsdlgo	0.43	0.79

Project	Mean	Standard deviation
xibocms	0.00	0.00
xroad-catalog	3.56	5.10
zato	13.83	14.90
Zero-K-Infrastructure	4.54	6.00
zuora	3.18	12.00

APPENDIX C – STATISTICS FOR CONTRACTS MODIFIED PER COMMIT

Project	Mean	Standard deviation
acplugins	2.94	1.76
alloc	1.00	0.00
animated-batman	1.55	1.04
apache-ode	1.37	1.17
apollo	2.43	3.06
battleship	2.40	0.89
bearded-archer	2.63	1.19
betsy	1.33	0.49
BPEL-splitting	2.13	1.55
bpmlabs-services	1.00	0.00
btc4j-ws	1.00	0.00
bypass-stored-value	1.00	0.00
c3pr	1.34	1.26
caaers	1.42	1.39
caarray	3.96	5.38
cagrid	1.55	1.33
cagrid-core	1.56	0.75
cagrid-grid-incubation	1.88	1.32
CalendarPortlet	1.25	0.71
CallCenter	2.75	1.28
camel	1.22	0.66
capture-hpc	1.00	0.00
carbon-analytics-common	2.85	0.91
carbon-apimgt	8.19	10.77
carbon-business-messaging	4.50	2.51
carbon-business-process	2.90	2.01

Project	Mean	Standard deviation
carbon-data	2.00	0.00
carbon-deployment	3.93	3.02
carbon-event-processing	2.93	2.07
carbon-identity	3.78	4.39
carbon-identity-framework	3.83	1.60
carbon-metrics	2.73	0.46
carbon-storage-management	3.00	0.00
cartoweb3	1.00	0.00
centreon-engine	1.00	0.00
channelAdvisorAccess	4.63	2.45
corral	1.32	0.67
csla	1.86	0.77
cws-esolutions	1.53	0.52
CxTest	1.00	0.00
dblog	1.00	0.00
DemoGames	2.67	0.52
DistMM	1.00	0.00
DL-Learner	2.19	1.17
doms-bitstorage	2.19	1.36
doms-server	1.57	0.96
droolsjbpm-integration	1.47	1.23
dssp-client	18.00	1.00
ebmsadapter	1.00	0.00
EDLProvider	1.70	0.67
EECloud	2.82	1.08
eucalyptus	1.00	0.00
eve-intel-map	2.83	0.75

Project	Mean	Standard deviation
ezags-xsd	2.71	2.05
gds-mis	3.42	2.23
GNDMS	3.79	4.14
gsn	5.50	1.22
Habitat	1.00	0.00
HandHeldInventory	1.20	0.45
helium	2.00	0.00
hotell	1.00	0.00
htcondor	1.54	0.81
idecore	2.44	1.24
identity-inbound-auth-oauth	4.00	0.00
ihub	1.94	0.77
incubator-stratos	5.18	5.34
IQ-Champions	2.33	0.82
iws	3.32	2.28
jdk7u-jdk	1.31	0.48
JoinToPlayClient	2.50	1.29
juddi	2.00	2.00
kask-kiosk	2.91	0.95
kc	4.13	12.89
kc.preclean	4.19	12.99
kdepim-ktimetracker-akonadi	1.67	1.19
kdepim-noakonadi	1.67	1.19
kopete	1.00	0.00
Liltarp-Assignment	2.56	0.53
lime-security-powerauth	1.50	0.59
lolsoap	2.00	0.00

Project	Mean	Standard deviation
magento2	1.41	0.78
mcgssg1_servidor	1.00	0.00
MDW	1.83	0.78
MOOLGOSS	2.44	0.88
mplus-api-wsdl	1.00	0.00
mule-cookbook	2.33	0.78
mule-wss-soap-example	1.00	0.00
multidatabase	1.91	0.83
myPress	2.71	0.49
NE-HSCIE-Core	1.05	0.22
NOTES-WEB	1.00	0.00
OCHP	1.29	0.89
ode	1.41	1.22
opencover	1.78	0.67
openspecimen	3.00	4.63
otrs-gitimport-test	1.00	0.00
oxalis	1.25	0.44
petals-se-activiti	1.89	1.03
petnet-web	1.00	0.00
php-amadeus	14.45	22.56
PizzaWaiter	3.08	1.12
polymony	2.00	0.00
processmaker	1.05	0.22
ProcessManager	1.06	0.25
python-lemonway	1.00	0.00
rig-client	1.00	0.00
RSB	1.70	1.22

Project	Mean	Standard deviation
s2	1.00	0.00
scape	1.05	0.23
scheduling-server	1.05	0.22
sci-flex-synapse-esper-plugin	4.00	0.00
sdk-client-tools-protex	30.14	33.36
SEServerExtender	6.86	5.73
SevenUpdate	3.55	1.78
simias	1.27	0.46
sipXtapi	1.03	0.16
sones	1.00	0.00
staff	1.11	0.33
steve	1.50	0.53
stratos	2.16	0.94
sts	1.33	0.48
SwingDriver	2.00	0.00
Sync	3.92	2.68
sysmgrt	2.25	1.16
taverna-grid	3.50	1.84
tempo	1.02	0.14
Tennis-Middleware	1.11	0.33
Thesis	2.57	0.53
UltraMarket-System	1.83	0.41
UMProVolleyWeb	1.00	0.00
weblabdeusto	1.06	0.24
Websitepanel	2.00	0.71
wizard	1.71	1.08
wsdlgo	1.17	0.41

Project	Mean	Standard deviation
xibocms	1.00	0.00
xroad-catalog	1.13	0.35
zato	1.00	0.00
Zero-K-Infrastructure	1.23	0.85
zuora	3.12	0.60