SOFTENG 2023

The Ninth International Conference on Advances and Trends in Software Engineering

ISBN: 978-1-68558-042-1

April 24th – 28th, 2023

Venice, Italy

SOFTENG 2023 Editors

Luigi Lavazza, Università degli Studi dell’Insubria, Italy

SOFTENG 2023

Forward

The Ninth International Conference on Advances and Trends in Software Engineering (SOFTENG 2023), held between April 24th and April 28th, 2023, continued a series of events focusing on these challenging aspects for software development and deployment, across the whole life-cycle.

Software engineering exhibits challenging dimensions in the light of new applications, devices, and services. Mobility, user-centric development, smart-devices, e-services, ambient environments, e-health and wearable/implantable devices pose specific challenges for specifying software requirements and developing reliable and safe software. Specific software interfaces, agile organization and software dependability require particular approaches for software security, maintainability, and sustainability.

We take here the opportunity to warmly thank all the members of the SOFTENG 2023 technical program committee, as well as all the reviewers. The creation of such a high-quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and effort to contribute to SOFTENG 2023. We truly believe that, thanks to all these efforts, the final conference program consisted of top-quality contributions. We also thank the members of the SOFTENG 2023 organizing committee for their help in handling the logistics of this event.

We hope that SOFTENG 2023 was a successful international forum for the exchange of ideas and results between academia and industry and for the promotion of progress in the field of software engineering.

SOFTENG 2023 Chairs

SOFTENG 2023 Steering Committee

Zeeshan Ali Rana, NUCES, Lahore, Pakistan

SOFTENG 2023 Publicity Chairs

Laura Garcia, Universitat Politecnica de Valencia, Spain
Javier Rocher Morant, Universitat Politecnica de Valencia, Spain

SOFTENG 2023
Committee

SOFTENG 2023 Steering Committee

Zeeshan Ali Rana, NUCES, Lahore, Pakistan

SOFTENG 2023 Publicity Chairs

Laura Garcia, Universitat Politecnica de Valencia, Spain
Javier Rocher Morant, Universitat Politecnica de Valencia, Spain

SOFTENG 2023 Technical Program Committee

Khelil Abdelmajid, Landshut University of Applied Sciences, Germany
Mo Adda, University of Portsmouth, UK
Bestoun S. Ahmed, Karlstad University, Sweden
Issam Al-Azzoni, Al Ain University of Science and Technology, UAE
Mahmoud Alfadel, Concordia University, Montreal, Canada
Vahid Alizadeh, College of Computing & Digital Media - DePaul University, USA
Washington Almeida, Cesar School | Center of Advanced Studies and Systems of Recife, Brazil
Hussein Almulla, University of South Carolina, USA / University of Anbar, Irak
Vu Nguyen Huynh Anh, Université Catholique de Louvain, Belgium
Pablo O. Antonino, Fraunhofer IESE, Germany
Darlan Arruda, University of Western Ontario, Canada
Jocelyn Aubert, Luxembourg Institute of Science and Technology (LIST), Luxembourg
Lerina Aversano, University of Sannio, Italy
Ali Babar, University of Adelaide, Australia
Doo-Hwan Bae, Software Process Improvement Center - KAIST, South Korea
Mohamed Basel Almourad, College of Technological Innovation - Zayed University, Dubai, UAE
Imen Ben Mansour, University of Manouba, Tunisia
Maya Benabdelhafid, Ecole Supérieure de Comptabilité et de Finances (ESCF) de Constantine, Algeria
Marciele Berger, University of Minho, Portugal
Marcello M. Bersani, Politecnico di Milano, Italy
Anna Bobkowska, Gdansk University of Technology, Poland
Antonio Brogi, University of Pisa, Italy
Azahara Camacho, RTI - Real Time Innovations, Spain
Qinglei Cao, University of Tennessee, Knoxville, USA
José Carlos Metrôlho, Polytechnic Institute of Castelo Branco, Portugal
Pablo Cerro Cañizares, Universidad Complutense de Madrid, Spain
Allaoua Chaoui, University Constantine 2 - Abdelhamid Mehri, Algeria
Andrea D'Ambrogio, University of Rome Tor Vergata, Italy
Lilian Michele da Silva Barros, Instituto Tecnológico de Aeronáutica, Brazil
Luciano de Aguiar Monteiro, Institute of Higher Education iCEV - Teresina-Piauí, Brazil
Amleto Di Salle, University of L'Aquila, Italy
Sigrid Eldh, Ericsson AB, Sweden
Gencer Erdogan, SINTEF Digital, Norway

Table of Contents

Lightweight Sample Code Recommendation System to Support Programming Education

Yoshihisa Udagawa
Faculty of Informatics, Tokyo University of Information Sciences
Chiba-city, Chiba, Japan
e-mail: yu207233@rsch.tuis.ac.jp

***Abstract*— One effective way to learn programming techniques is to refer to sample programs. As the number of sample programs increases, however, it becomes difficult and time-consuming to find appropriate sample code visually. To overcome this shortcoming, research and development of program recommendation systems have been actively conducted. This paper discusses a recommendation system for Java sample programs using an unsupervised machine learning technique. The proposed system includes three major steps: (1) extracting invoked methods used in each sample program, (2) clustering the sample programs by applying a data mining technique to the extracted methods, and (3) ranking the programs by calculating a weighted average of the extracted methods. Experiments using file input and output sample programs indicate that the proposed system has sufficient potential to support programming education.**

Keywords—Recommendation System for Software Engineering; Mining Software Repository; Maximal Frequent Itemset; Tf-idf; Unsupervised Machine Learning; Programming Education.

I. INTRODUCTION

Sample programs are an important source for learning new programming technologies. In particular, sample programs for using Application Programming Interfaces (API) related to open-source programs are available on the Internet. Since the amount of publicly concerning sample code becomes enormous, it might become time-consuming and error-prone to find appropriate sample code visually. Over the past few decades, there has been a great deal of research and development on the systems that provide useful programming information for students and developers.

Recommendation systems are generally employed in online stores and video/music websites, where rankings of items are calculated based on users' reactions and similarities among products and/or works. The recommendation system for software development deals with artifacts, such as sample programs, specifications, test cases and bug reports. Several techniques have been developed to collect, rank, and visualize similar artifacts based on various indicators reflecting their nature. These techniques are often specific to software engineering and cause a recommendation system to be called a Recommendation System for Software Engineering (RSSE) [1].

Gasparic and Janes [2] survey 46 research and development articles on RSSE published between 2003 and 2013, and categorize them with respect to the covered data and the methods for recommendation. The most common type of covered data is source code with 21 papers, followed by help information to perform changes with 6 papers. As for ranking method, list format is the most common with 33 papers, followed by document format with three papers, and table format with two papers.

Hsu and Lin [3] propose a recommendation system based on frequent patterns in source code. They originally define 17 syntax patterns and extract them from the source code under study. A sequence pattern extraction algorithm based on frequency known as *Prefix-Span* is applied to recommend API usage patterns.

Katirtzis, Diamantopoulos, and Sutton [4] discuss an algorithm that extracts API call sequences and then clusters them to create an API usage summary known as a source code snippet. Hierarchical clustering is performed by calculating the distance of extracted API call sequences using the longest common subsequence algorithm. Then, code slice techniques are applied to create a source code snippet.

Diamantopoulos and Symeonidis [5] develop a system to recommend sample code stored in software repositories on the Internet, such as GitHub, GitLab and Bitbucket. The input to the system is a code fragment presented by a user, and the output is a set of sample codes similar to the code fragment. Similarities among source codes are calculated based on the vector space model and the Levenshtein distance.

Hora [6] discusses a source code recommendation system that analyzes source code contained in a particular project and creates ranked API usage examples on a web site. The system ranks the source code based on three quality measures, i.e., similarity, readability, and reusability. The similarity is calculated using the cosine similarity in data analysis, while readability and reusability are calculated using indicators developed in software engineering studies.

Nguyen, Rocco, Sipio, Ruscio, and Penta [7] implement a system to present API usage in a timely manner during a coding process, and discuss the evaluation of experimental results. The system calculates the similarity among similar projects by Term Frequency-Inverse Document Frequency *(tf-idf)* [8] and ranks API usage patterns using a collaborative filtering technique [9].

This paper discusses a lightweight recommendation system for analyzing Java sample programs that are collected from the Internet. The system clusters Java sample programs based on the methods that are invoked by the programs so that each cluster represents a programming subject. The system ranks the sample programs using a *tf-idf* weighted vector space model for each clustering. Since the higher ranked samples contain more invoked methods than those ranked lower, this system assists a student in selecting sample code suitable for learning.

The contributions of this study are as follows:

I. In general, method call patterns differ from one programming subject to another. This system can automatically cluster sample programs by programming subjects and represent them to students.

II. The RSSEs proposed so far employ hard-clustering, if any. In hard-clustering, the results depend on the initial values and have the restriction that one sample belongs to only one cluster. This study employes soft-clustering. Therefore, a sample program can belong to multiple clusters, and a cluster only contains related programs.

III. By modifying *tf-idf* to give greater weights to the methods that frequently appear in a cluster, sample programs that fit the subject of a cluster and invoke many rare methods are ranked higher.

IV. The proposed system employs unsupervised machine learning, making it lightweight to use, operate and maintain the system.

The remainder of the paper is organized as follows. Section II gives the architecture of the proposed system. Section III describes the implementation of the main functions of the proposed system. Section IV shows the experimental results using file I/O sample programs. Section V discusses other implementation options. Section VI concludes the paper with our plans for future work.

II. Overview of Proposed System

This section describes the architecture of the proposed system from the functional point of view, and outlines typical usage.

A. *Code Analyzer*

Figure 1 depicts the architecture of the proposed system. The input for this system is sample programs available on the Internet. Currently, sample programs are collected manually and stored in a specific project typically in *Eclipse*, an Integrated Development Environment (IDE) for Java [10]. In this study, we assume that all sample programs are correct and work properly.

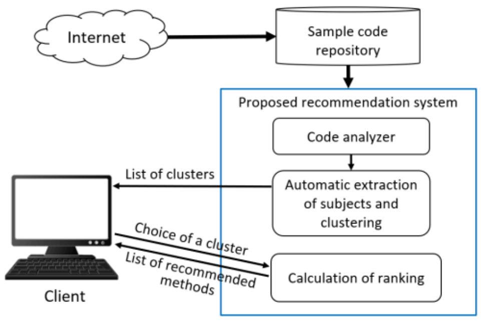


Figure 1. Overview of the proposed system.

Figure 2 shows a set of sample programs used in this study, which is stored in a project named *Sample_File_IO* in *Eclipse*. The sample program can be stored in packages. There is no limitation to the depth of the package hierarchy. As discussed later sections, these programs are concerned with binary and string file I/O. Programs can be stored in any directory other than an *Eclipse* project.

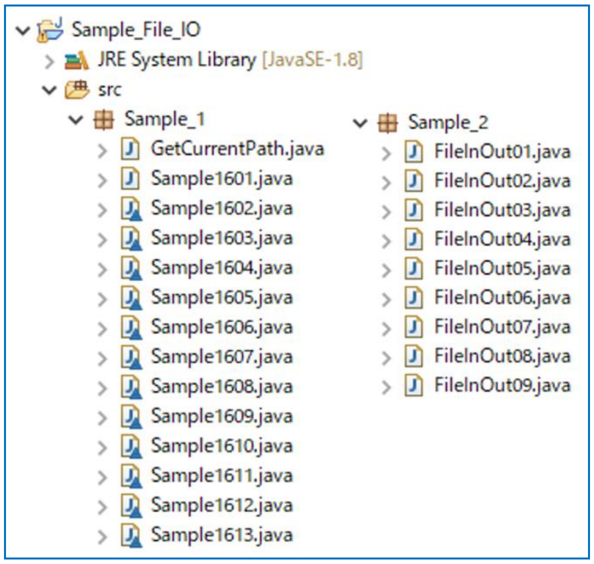
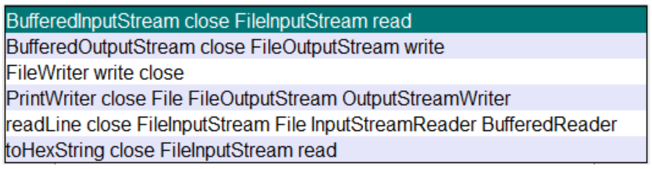

```
Sample_File_IO
  JRE System Library [JavaSE-1.8]
  src
    Sample_1                      Sample_2
      GetCurrentPath.java           FileInOut01.java
      Sample1601.java               FileInOut02.java
      Sample1602.java               FileInOut03.java
      Sample1603.java               FileInOut04.java
      Sample1604.java               FileInOut05.java
      Sample1605.java               FileInOut06.java
      Sample1606.java               FileInOut07.java
      Sample1607.java               FileInOut08.java
      Sample1608.java               FileInOut09.java
      Sample1609.java
      Sample1610.java
      Sample1611.java
      Sample1612.java
      Sample1613.java
```

Figure 2. Sample programs stored in *Eclipse* project.

The *code analyzer* in Figure 1 extracts method declarations and invoked method names from all Java files under the specified directory or project. A list of method names being invoked is used for clustering the declared methods and ranking them.

B. *Automatic Identification of Subjects and Clusters*

Following code analysis, the *Apriori* algorithm [11] is started to identify the set of invoked methods that occur frequently. Based on the frequent method set, programming subjects are automatically identified. Each subject corresponds to a cluster. Figure 3 shows an example of identified clusters.



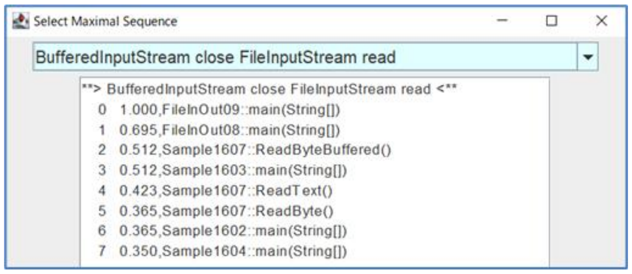
BufferedInputStream close FileInputStream read
BufferedOutputStream close FileOutputStream write
FileWriter write close
PrintWriter close File FileOutputStream OutputStreamWriter
readLine close FileInputStream File InputStreamReader BufferedReader
toHexString close FileInputStream read

Figure 3. Identified programming subjects and clusters.

This study uses a soft-clustering technique based on a maximal frequent itemset [12], i.e., a compact itemset that represents a frequent itemset. Strictly, the method names displayed in each cell of the combo box in Figure 3 are elements of a maximal frequent itemset. For example, "*BufferedInputStream close FileInputStream read"* suggests from the method names that the cluster is related to the subject of reading binary data.

C. *Calculation of Recommended Ranking*

Selecting a cell in the combo box in Figure 3 causes to specify a cluster of methods, which starts calculations of recommendation values for each of the declared methods in the cluster. Figure 4 shows an example of a method recommendation. The values of recommendation for each declared method are normalized so that the maximum value is equal to one.



```
BufferedInputStream close FileInputStream read
**> BufferedInputStream close FileInputStream read <**
 0  1.000,FileInOut09::main(String[])
 1  0.695,FileInOut08::main(String[])
 2  0.512,Sample1607::ReadByteBuffered()
 3  0.512,Sample1603::main(String[])
 4  0.423,Sample1607::ReadText()
 5  0.365,Sample1607::ReadByte()
 6  0.365,Sample1602::main(String[])
 7  0.350,Sample1604::main(String[])
```

Figure 4. Sample of program recommendation.

Method names are prefixed with class names, so that a student can easily check method source code using an IDE, such as *Eclipse, NetBeans and IntelliJ IDEA*.

III. IMPLEMENTATION

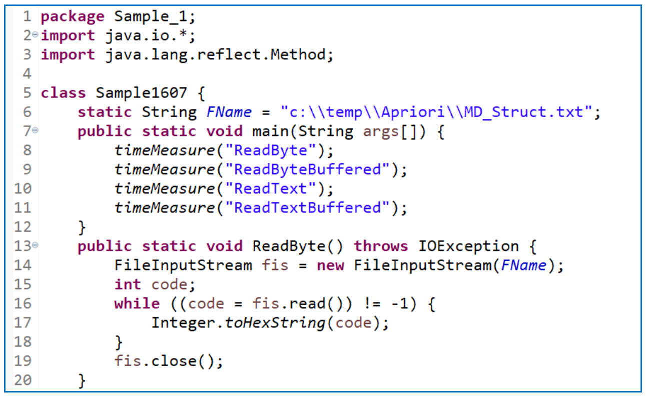
This section describes the implementation of three major steps of the proposed system. Those steps are code analysis, clustering, and ranking.

A. *Code Analysis for Extracting Invoked Method Set*

Functions necessary for system development are typically provided as runtime methods in Java. After learning the control structure of programs and object-oriented techniques, students and developers enhance their programming skills by learning how to use the runtime methods provided by Java communities. Therefore, the methods being invoked are closely related to the functionality of the program. In this study, we make the assumption that program similarity can be computed by the similarity of the method sets being invoked.

The *code analyzer* in Figure 1 extracts a declared method signature and a set of invoked methods. We implemented the code analyzer using the *Scanner* class [13], a tokenizer in *Eclipse* Java Development Tools (JDT) core. This class provides the ability to classify the tokens in a Java program into more than 100 types, and excludes comments allowing efficient analysis of executable statements. The class is widely used in *Eclipse* for navigating Java programs, including a class-method hierarchy.

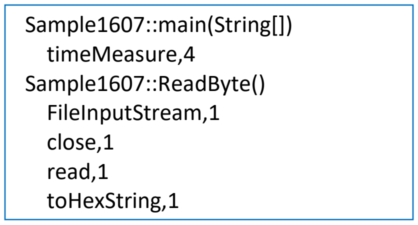
Figure 5 shows a sample of a Java program. Figure 6 shows a list of declared methods and invoked ones that are generated from the Java program. A method with the same name is usually invoked multiple times in a declared method. Therefore, the analyzer extracts the method name and the number of times invoked, which are used for calculating cosine similarity. For example, the *main()* method in the *Sample1607* class in Figure 5 invokes the *timeMeasure()* method four times.



```
package Sample_1;
import java.io.*;
import java.lang.reflect.Method;

class Sample1607 {
    static String FName = "c:\\temp\\Apriori\\MD_Struct.txt";
    public static void main(String args[]) {
        timeMeasure("ReadByte");
        timeMeasure("ReadByteBuffered");
        timeMeasure("ReadText");
        timeMeasure("ReadTextBuffered");
    }
    public static void ReadByte() throws IOException {
        FileInputStream fis = new FileInputStream(FName);
        int code;
        while ((code = fis.read()) != -1) {
            Integer.toHexString(code);
        }
        fis.close();
    }
```

Figure 5. Sample Java programs.



```
Sample1607::main(String[])
  timeMeasure,4
Sample1607::ReadByte()
  FileInputStream,1
  close,1
  read,1
  toHexString,1
```

Figure 6. Extracted method names and the number of times invoked.

It should be noted that the methods, such as *println()* and *printStackTrace()*, are intentionally excluded from the extraction process because they are often used to print data values for debugging purpose and fail to characterize the function of a declared method.

B. *Apriori Algorithm for Identifying Subjects and Clusters*

Apriori algorithm proposed by Agrawal and Srikant [11] starts by identifying the frequent individual items and extending them to larger itemset as long as those itemset frequently appear in the database under consideration.

Let a database D be a set of transactions t, i.e., $D= \{t_1, t_2,\dots, t_n\}$. Let each transaction t_i be a nonempty set of itemset, i.e., $t_i = \{i_{i1}, i_{i2},\dots, i_{im}\}$. The itemset is a nonempty set of items observed together.

A support value of an itemset refers to the number of transactions that contain the itemset. In terms of D and t_i, the support value of an itemset X is defined by the following formula:

$$Support(X)= |\{ t_i \in D : X \subseteq t_i \ \& \ 1 \leq i \leq n \}| \qquad (1)$$

A set of items is called frequent if its support value is greater than a user-specified minimum support value, i.e., *minSup*.

Here, we cite the *Apriori* principle:

If an itemset is frequent, then all of its subsets are also frequent.

This means that if a set is infrequent, then all of its supersets are infrequent. The *Apriori* algorithm works based on this principle, in which k-frequent item sets are utilized to identify $k+1$ frequent item sets.

Since the frequent itemset generated by the *Apriori* algorithm tends to be very large, it is beneficial to identify a compact representation of all the frequent itemset for a particular database. One such approach is to use a maximal frequent itemset [12].

Definition:
A maximal frequent itemset is a frequent itemset for which none of its immediate supersets are frequent.

By definition, all frequent itemset can be derived from the set of maximal itemset. Table I shows an example of a database consisting of five transactions of itemset. Figure 8 illustrates an example of the maximal frequent itemset in a lattice structure where a node corresponds to an itemset and arcs correspond to the subset relation [12]. *MinSup* is set to 1 or 20% (= 1/5*100).

TABLE I. EXAMPLE OF DATABASE

Transaction ID	Item set
1	A B
2	A D
3	B C
4	A C D
5	B C D

In Figure 7, the nodes surrounded by solid lines indicate the frequent itemset, while the nodes with yellow backgrounds indicate the maximal frequent itemset.

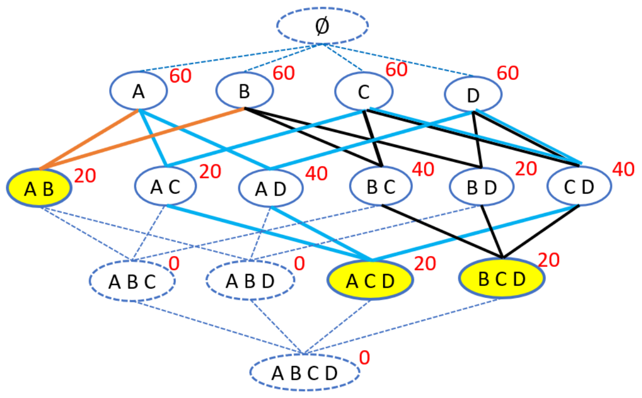


Figure 7. Maximal frequent itemset in lattice structure.

The frequent itemset can be soft-clustered by the maximal frequent itemset. For example, the subsets of {A, C, D} are {A, C}, {A, D}, {C, D}, {A}, {C}, {D}. The subsets of {A, B} are {A}, {B}. Analogously, the subsets generated from {B, C, D} are {B, C}, {B, D}, {C, D}, {B}, {C}, {D}. These subsets formulate three soft-clusters sharing the subsets, such as {A}, {B}, {C}, {D}.

Table II shows the number of the frequent itemset, the maximal frequent itemset, and the compression ratio of when the minimum support *minSup* varies. The experiment is performed using 33 methods declared in 23 Java files. The number of unique invoked methods is 36.

TABLE II. NUMBER OF ELEMENTS IN ITEMSET

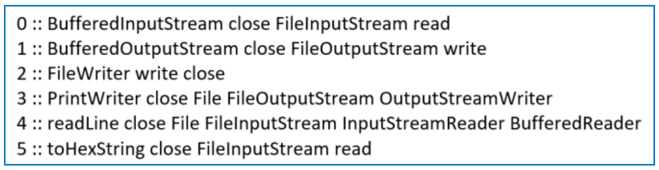
MinSup	No. of frequent itemset	No. of maximal frequent itemset	Ratio (%)
6%	236	11	4.7
8%	236	11	4.7
10%	131	7	5.3
12%	127	6	4.7
14%	127	6	4.7
16%	29	7	24.1
18%	25	6	24.0
20%	25	6	24.0
22%	15	3	20.0
24%	13	4	30.8
26%	13	4	30.8
28%	11	5	45.5
30%	11	5	45.5
32%	11	5	45.5
34%	10	5	50.0
36%	3	1	33.3

For example, when *minSup* is 12%, the number of frequent itemset is 127 and the number of maximal frequent itemset is 6. The number of elements compresses to 4.7%. The maximal frequent itemset headlines the frequent itemset and defines the cluster.

C. Clustering Methods Using Maximal Frequent Itemset

More than ten binary programs that implement the *Apriori* algorithm are available on the web page maintained by Borgelt [14]. For the sake of openness and efficiency of implementation, this study uses *fpgrowth.exe* listed on the web page. Specifically, we implement a maximal-frequent-itemset generating function by calling *fpgrowth.exe* using *java.lang.Runtime.exec()* that executes the specified command and arguments in a separate process. The input data for this program is the set of invoked methods for each declared method shown in Figure 6, ignoring the number of invoked methods citations.

Figure 8 shows the maximal frequent itemset obtained from the sample program shown in Figure 2, with a *minSup* of 11%. The maximal frequent itemset corresponds to the programming subjects and is shown in Figure 3 as well.



```
0 :: BufferedInputStream close FileInputStream read
1 :: BufferedOutputStream close FileOutputStream write
2 :: FileWriter write close
3 :: PrintWriter close File FileOutputStream OutputStreamWriter
4 :: readLine close File FileInputStream InputStreamReader BufferedReader
5 :: toHexString close FileInputStream read
```

Figure 8. Example of generated maximal frequent itemset.

Figure 9 shows a list of declared methods that contain at least two invoked method names that are elements of a maximal frequent itemset. These clusters are broadly classified into two categories, i.e., those related to reading files and those related to writing files.

```
0  BufferedInputStream close FileInputStream read
   FileInOut08::main(String[])
   FileInOut09::main(String[])
   Sample1602::main(String[])
   Sample1603::main(String[])
   Sample1604::main(String[])
   Sample1607::ReadByte()
   Sample1607::ReadByteBuffered()
   Sample1607::ReadText()
1  BufferedOutputStream close FileOutputStream write
   FileInOut07::main(String[])
   FileInOut09::main(String[])
   Sample1608::main(String[])
   Sample1609::main(String[])
   Sample1610::main(String[])
   Sample1613::byteBufferedWrite()
   Sample1613::byteWrite()
2  FileWriter write close
   Sample1611::main(String[])
   Sample1612::main(String[])
   Sample1613::textBufferedWrite()
   Sample1613::textWrite()
3  PrintWriter close File FileOutputStream OutputStreamWriter
   FileInOut01::main(String[])
   FileInOut02::main(String[])
   FileInOut03::main(String[])
   FileInOut06::main(String[])
   FileInOut07::main(String[])
   FileInOut09::main(String[])
   Sample1610::main(String[])
4  readLine close File FileInputStream InputStreamReader BufferedReader
   CountUniqueMathod::main(String[])
   FileInOut04::main(String[])
   FileInOut05::main(String[])
   FileInOut06::main(String[])
   FileInOut08::main(String[])
   FileInOut09::main(String[])
   GetCurrentPath::main(String[])
   Sample1604::main(String[])
   Sample1607::ReadText()
5  toHexString close FileInputStream read
   FileInOut08::main(String[])
   FileInOut09::main(String[])
   Sample1602::main(String[])
   Sample1603::main(String[])
   Sample1604::main(String[])
   Sample1607::ReadByte()
   Sample1607::ReadByteBuffered()
   Sample1607::ReadText()
```

Figure 9. Methods belonging to each cluster.

Due to soft-clustering, one method belongs to multiple clusters. For example, *Sample1607::ReadByte()* is included in clusters 0 and 5, and *Sample1607::ReadText()* is included in clusters 0, 4 and 5.

D. Calculation of Recommendation Ranking

1) Definition of tf-idf

The Term Frequency-Inverse Document Frequency *(tf-idf)* weight [8] is one that commonly used in information retrieval. In the context of our study, the *tf-idf* can be rephrased as follows:

Tf (term frequency) means the frequency of an invoked method name in a sample program,

Idf (inverse document frequency) indicates a numerical value that reflects how rare or important an invoked method name in a set of sample programs.

Among several options to calculate the *tf* and *idf*, we adopt the following definitions.

Tf_i is defined as the number of occurrences of an invoked method *i*.

Idf_i is defined as $log(N/DF_i)$, where *N* is the total number of declared methods that occur in a set of sample programs, and DF_i is the number of declared methods where an invoked method *i* appears at least once. It should be noted that idf_i of an invoked method *i* that appears in all declared methods is equal to $log(N/N)$, which is equal to 0.

2) Calculating Tf-idf for Sample Program Recommendation

As mentioned earlier, the maximal frequent itemset consists of a set of method names that suggest programming subjects. The maximal frequent itemset is displayed on the combo box in the GUI as shown in Figure 3. The proposed system identifies a set of declared methods when a user selects a cell on the combo box, and then starts to compute *tf* and *idf* for the set of declared methods. Table III lists the *tf* and *idf* values of the invoked method names corresponding to the maximal frequent itemset *{BufferedInputStream, close, FileInputStream, read}* that is shown at the top of Figure 9. There are 19 invoked methods in the sample programs related to the maximal frequent itemset.

TABLE III. *TF* AND *IDF* VALUES OF INVOKED METHOD NAMES

No.	Invoked method name	Tf	Idf
0	BufferedInputStream	4	0.574
1	BufferedOutputStream	1	1.176
2	BufferedReader	6	0.398
3	File	6	0.398
4	FileInputStream	12	0.097
5	FileOutputStream	2	0.875
6	FileReader	3	0.699
7	InputStreamReader	6	0.398
8	OutputStreamWriter	1	1.176
9	PrintWriter	1	1.176
10	close	15	0
11	flush	1	1.176
12	format	1	1.176
13	length	1	1.176
14	read	11	0.135
15	readLine	4	0.574
16	toChars	2	0.875
17	toHexString	4	0.574
18	write	1	1.176

Since the proposed system uses clustering based on a maximal frequent itemset, the method names that are included in the Maximal Frequent Itemset (MFI) should be considered to characterize the sample programs more strongly than the others. In this study, the weights of the invoked method names are adjusted using the following formula.

Let *MFI* be the maximal frequent itemset specified by a user and idf_{max} be the maximum *idf* values.

$$\begin{array}{ll} Adjusted\ idf_j = idf_j + idf_{max} & \text{if } j \in MFI \\ \qquad\qquad\quad = idf_j & \text{if } j \notin MFI \end{array} \qquad (2)$$

Table IV shows the adjusted *idf* values for the maximal frequent itemset *{BufferedInputStream, close, FileInputStream, read}*.

TABLE IV. ADJUSTED *IDF* VALUES

Invoked method name	Tf	Idf	Adjusted idf
BufferedInputStream	4	0.574	1.75
FileInputStream	12	0.097	1.273
close	15	0	1.176
read	11	0.135	1.311

The degree of recommendation $DegR_i$ for a declared method *i* is calculated as:

$$DegR_i = \sum_{k=0}^{k=M-1} tf_{ik} * (Adjusted\ idf_k) \qquad (3)$$

where tf_{ik} is the number of occurrences of the invoked method *k* in the declared method *i*, and idf_k is the inverse document frequency of the invoked method *k*.

Table V shows the degrees of recommendation for the declared method related to the maximal frequent itemset *{BufferedInputStream, close, FileInputStream, read}*.

TABLE V. DEGREES OF RECOMMENDATION FOR SAMPLE PROGRAMS

No.	DegR	Name of sample program
0	8.26	FileInOut08::main(String[])
1	11.885	FileInOut09::main(String[])
2	4.334	Sample1602::main(String[])
3	6.084	Sample1603::main(String[])
4	4.158	Sample1604::main(String[])
5	4.334	Sample1607::ReadByte()
6	6.084	Sample1607::ReadByteBuffered()
7	5.033	Sample1607::ReadText()

The maximal degree of recommendation is normalized to be 1 and displayed in the GUI. For the lists in Table 3, the normalized degrees of recommendation are obtained by dividing all the degrees by 11.885. This calculation generates the final list of recommendations shown in Figure 4.

Performance is measured 10 times for the following two processes that comprise this system. Both include the time displayed in the GUI.

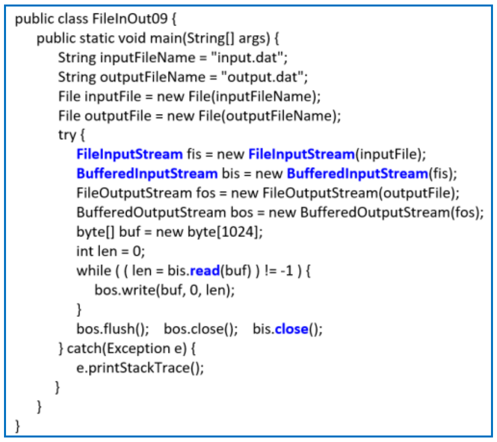
(1) From the start of parsing to the end of clustering: average 360.8ms, standard deviation 10.5ms

(2) After specifying a cluster to generating a list of recommendations: average 128.6ms, standard deviation 5.95ms

The specifications of a PC used are as follows:
CPU: AMD Ryzen 7 5700U (Laptop PC)
RAM: 16.0 GB
OS: Windows 10 Home 64 bit.

IV. EXPERIMENTAL RESULTS

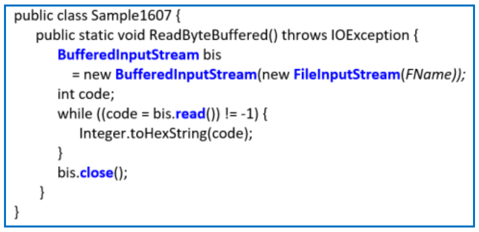
This section describes experimental results. Figure 10 shows the sample program or the declared method that corresponds to the top of the recommended list in Figure 4 with a normalized recommendation value of 1.000. The sample program includes all of the invoked methods that constitute the maximal frequent itemset *{BufferedInputStream, close, FileInputStream, read}*. In addition, it contains essential methods for binary file outputs, e.g., *FileOutputStream(), write()*.



```
public class FileInOut09 {
    public static void main(String[] args) {
        String inputFileName = "input.dat";
        String outputFileName = "output.dat";
        File inputFile = new File(inputFileName);
        File outputFile = new File(outputFileName);
        try {
            FileInputStream fis = new FileInputStream(inputFile);
            BufferedInputStream bis = new BufferedInputStream(fis);
            FileOutputStream fos = new FileOutputStream(outputFile);
            BufferedOutputStream bos = new BufferedOutputStream(fos);
            byte[] buf = new byte[1024];
            int len = 0;
            while ( ( len = bis.read(buf) ) != -1 ) {
                bos.write(buf, 0, len);
            }
            bos.flush();   bos.close();   bis.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure 10. Sample program with recommendation value 1.000.

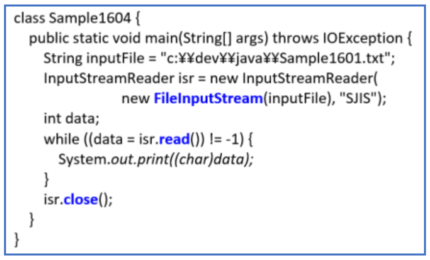
Figure 11 shows a sample program with a normalized recommendation value of 0.512. It contains all four method names that constitute the maximal frequency itemset and another method, i.e., *toHexString()*. Compared to the sample program in Figure 10, this provides a concise one.



```
public class Sample1607 {
    public static void ReadByteBuffered() throws IOException {
        BufferedInputStream bis
          = new BufferedInputStream(new FileInputStream(FName));
        int code;
        while ((code = bis.read()) != -1) {
            Integer.toHexString(code);
        }
        bis.close();
    }
}
```

Figure 11. Sample program with recommendation value 0.512.

Figure 12 shows a sample program with a normalized recommendation value of 0.350. It includes three method names of the maximal frequency itemset and another method, i.e., *InputStreamReader()*.



```
class Sample1604 {
    public static void main(String[] args) throws IOException {
        String inputFile = "c:¥¥dev¥¥java¥¥Sample1601.txt";
        InputStreamReader isr = new InputStreamReader(
                    new FileInputStream(inputFile), "SJIS");
        int data;
        while ((data = isr.read()) != -1) {
            System.out.print((char)data);
        }
        isr.close();
    }
}
```

Figure 12. Sample program with recommendation value 0.350.

Compared to the sample programs in Figures 10 and 11, Figure 12 shows the most concise program regarding a method usage for file read operations. These results demonstrate that the proposed system works as expected.

V. Discussion

A. Syntax Analysis

In this study, the *Scanner* [13] class is used for parsing sample programs mainly because it reduces development effort. There are several options of parsing tools, including *JavaParser* [15] and *ANTLR* [16], both of which generate an Abstract Syntax Tree (AST). AST is an intermediate representation of a program's source code in a tree structure. "Traversing" an AST that would require a few hundred lines of programming allows applications to perform more complex operations than a mere method name extraction. ANTLR can parse formal languages other than Java. All parsing tools work independently of IDEs and can parse sample code stored in arbitrary directories.

B. ChatGPT

ChatGPT is a chat-based tool released by OpenAI in Nov 2022 [17]. The latest *ChatGPT Feb 13* version allows users to chat about Java sample code for File I/O successfully. However, the sample code is limited to what *ChatGPT* has already learned. Since a learning process is exclusively conducted by an OpenAI team, it is difficult for a lecturer to configure sample programs to fit her/his classes. The method proposed in this study allows the lecturer to compose sample programs tailored for a class, even if those programs are specific or even unusual.

VI. Conclusion and Future Work

This study deals with a recommendation system of sample programs using unsupervised machine learning. The proposed system soft-clusters the sample programs based on the set of invoked method names that frequently observed. The clustering corresponds to programming subjects and is performed automatically using the *Apriori* algorithm. The recommended ranking of the sample programs is calculated based on an adjusted *tf-idf* model that takes the method name and number of times it is invoked.

It is confirmed through experiments using file I/O sample programs that declared methods including useful information on the programming subjects, such as read and write string/binary data, are ranked in higher position. This result indicates that the proposed recommendation system has sufficient potential to support programming education.

The *Apriori* algorithm employed in this study requires the minimum number of supports, i.e., *minSup*, to be specified in advance. The ability to automatically determine the optimal *minSup* is left as a topic for future research. Manual collection of sample programs is a drawback of this study. Sample code downloader is an issue for future development. Additional experiments on larger sample programs are planned to verify the effectiveness of the proposed recommendation system for programming education.

References

[1] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, "Recommendation systems for software engineering," IEEE Software 27, pp. 80-86, Jul. 2010, DOI: 10.1109/MS.2009.161

[2] M. Gasparic and A. Janes, "What Recommendation Systems for Software Engineering Recommend: A Systematic Literature Review," Journal of Systems and Software 113, pp. 101-113, Mar. 2016, DOI: 10.1016/j.jss.2015.11.036

[3] S.-K. Hsu and S.-J. Lin, "Mining Source Codes to Guide Software Development," Asian Conference on Intelligent Information and Database Systems, pp. 445-454, Mar. 2010, DOI: 10.1007/978-3-642-12145-6_46

[4] N. Katirtzis, T. Diamantopoulos, and C. Sutton, "Summarizing Software API Usage Examples using Clustering Techniques," Proc. of the 21st International Conference on Fundamental Approaches to Software Engineering. vol. 10802, Springer, pp. 189-206, Apr. 2018, DOI: 10.1007/978-3-319-89363-1_11

[5] T. Diamantopoulos and A. Symeonidis, "Mining Source Code for Component Reuse," Mining Software Engineering Data for Software Reuse, Advanced Information and Knowledge Processing. Springer, pp. 133-174, Mar. 2020, DOI: 10.1007/978-3-030-30106-4_6

[6] A. Hora, "APISonar: Mining API usage examples," Wiley Online Library, Software: Practice and Experience Vol. 51, Issue 2, pp. 319-352, Oct. 2020, DOI: 10.1002/spe.2906

[7] P. T. Nguyen, J. D. Rocco, C. D. Sipio, D. D. Ruscio, and M. D. Penta, "Recommending API Function Calls and Code Snippets to Support Software Development," IEEE Transactions on Software Engineering, Vol. 48, Issue 7, pp. 2417-2438, Jul. 2022, DOI: 10.1109/TSE.2021.3059907

[8] G. Sidorov. "Vector Space Model for Texts and the tf-idf Measure," In Syntactic n-grams in Computational Linguistics, pp.11-15, Apr. 2019, Springer, Cham, ISBN: 978-3-030-14770-9.

[9] A. Roy, "Introduction to Recommender Systems-1: Content-Based Filtering and Collaborative Filtering," Available from: https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421 [retrieved: Jul. 2020]

[10] Eclipse foundation, "Download Eclipse Technology that is right for you," Available from: https://www.eclipse.org/downloads/ [retrieved: Mar. 2023]

[11] R. Agrawal and R. Srikant, "Mining sequential patterns," Proc. 11th IEEE International Conference on Data Engineering (ICDE), pp.3-14, 1995, DOI: 10.1109/ICDE.1995.380415

[12] J. Rousu, "Finding frequent itemsets - concepts and algorithms," University of Helsinki, Available from: https://www.cs.helsinki.fi/group/bioinfo/teaching/dami_s10/dami_lecture4.pdf [retrieved: Apr. 2010]

[13] IBM Rational Software Architect, "Interface IScanner," in org.eclipse.jdt.core.compiler, Available from: https://www.ibm.com/docs/ja/developer-for-zos/9.5.1?topic=SSQ2R2_9.5.1/org.eclipse.wst.jsdt.doc/reference/api/org/eclipse/wst/jsdt/core/compiler/IScanner.htm [retrieved: Mar. 2021]

[14] "Christian Borgelt's Web Pages," Available from: https://borgelt.net/fpgrowth.html [retrieved: Nov. 2022]

[15] JavaParser.org, "Tools for your Java code," Available from: https://javaparser.org [retrieved: 2019]

[16] T. Parr, "Download ANTLR", Available from: https://www.antlr.org/download.html [retrieved: Feb. 2023]

[17] OpenAI, "Introducing ChatGPT," Available from: https://openai.com/blog/chatgpt [retrieved: Nov. 2022]

Taxonomy of Requirements Specification Templates

Hiba Hnaini*, Raúl Mazo*, Paola Vallejo†, Jose Galindo‡, and Joël Champeau*
*Lab-STICC, ENSTA Bretagne, Brest, France
{hiba.hnaini, raul.mazo, joel.champeau}@ensta-bretagne.fr
†GIDITIC, Universidad EAFIT, Medellín, Colombia
pvallej3@eafit.edu.co
‡Dpto. de Lenguajes y Sistemas Inf., Universidad de Sevilla, Sevilla, Spain
jagalindo@us.es

***Abstract*—Requirements specification is an early stage of system design. It consists of rephrasing and documenting stakeholders' explanations and needs in the form of clear and coherent requirements. However, these requirements are often expressed in natural language since it is the easiest communication method. Researchers have proposed semi-structured natural language templates or boilerplates for specifying functional and non-functional requirements, which consider security requirements. This seeks to enhance the quality of the requirements specifications and simplify their transformation to system models. However, it is still unknown what concepts, quality attributes, and good practices should be considered to specify requirements in a semi-structured natural language and how that information has been considered in the existing templates. In this paper, we aim to determine how templates are related among them and what are the implications (e.g., complexity, completeness, time) of using one or another. In this paper, we identify each template's concepts, quality attributes, and good practices by studying the template's aspects and then using a running example to formulate requirements using these templates. We also identify the aspects repeated or inherited from one template to another. This paper puts forward a taxonomy of requirements specification templates that categorize and specify the sources of the templates, which helps determine what template considers the aspects of another.**

***Index Terms*—Security Requirements Template, Boilerplate, Requirements Specification, Natural Language, Taxonomy.**

I. Introduction

The increase of security threats on software and hardware systems within the past few years has imposed consideration of security at all stages of system development, starting from the requirements specification stage. However, eliciting precise and non-complex security requirements can take time and effort. This challenge originates from the need for guides that help security requirements engineers define their requirements. Similarly to standard requirements, security requirements also aim that (i) the same interpretation is reached by all readers and (ii) this interpretation corresponds to the idea that the author of the requirement was trying to convey.

According to Denger *et al.* [1], the most common method for specifying and documenting requirements is Natural Language (NL) since it needs no training and is within reach. However, the drawbacks of NL are too significant to disregard. As Dalpiaz *et al.* [2] explain, these disadvantages are ambiguity, unclarity, inconsistency, and incompleteness. Mavin *et al.* [3] add the disadvantages of vagueness, complexity, duplication, verbosity, implementation, and untestability. Several researchers have proposed the use of semi-structured natural language in the form of guidelines, templates, boilerplates, patterns, and so on to mitigate some of the weaknesses of natural language when writing requirements. For example, a template controls the structure of the requirement by possibilities and restrictions while preserving the advantage of being in NL. In addition, this method reduces faults in the early stages of a system's development process.

Some proposals consist of generic guidelines that make it possible to specify requirements for almost any type of systems [4] [5].

Other proposals present different strategies to facilitate the work of the requirements engineers and, therefore, improve the quality of the products specified by modeling textual requirements. For example, through standard reference data [6], a generic syntactic requirements specification template [7], a robust template [8], or a template based on the 5W1H (Why, Who, Where, When, What, How) questions [9].

Other researchers defined and used templates for a specific domain. For example, Esser and Struss propose a natural language template-based interface to acquire requirements for functional testing of control software for passenger vehicles [10], and Mavin *et al.* present a set of structural rules to address common requirements problems, such as ambiguity, complexity, and vagueness [3]. The rule set allows all requirements to be expressed in natural language in one of five simple templates. The rule set was applied to extract requirements for an aero engine control system from an airworthiness standard document, and Mahmud *et al.* propose a toolchain for structured requirements specification in the Requirements Specification and Analysis (ReSA) language. "ReSA is an ontology-based requirements specification language tailored to automotive embedded systems development" [11].

Other researchers concentrated on security requirements specifications. For example, Toval *et al.* present a method for eliciting and specifying system and software requirements, including a repository containing reusable requirements, a spiral process model, and a set of requirements document templates [12]. Firesmith discussed the value of reusable parameterized templates for specifying security requirements [13]. Firesmith outlined an asset-based risk-driven analysis approach for determining the appropriate actual parameters to use when reusing such parameterized templates to specify security requirements that should improve the quality of security requirements in requirements specifications. Kamalrudin *et al.* propose a security requirements library and template to assist the requirements engineer in writing security requirements by providing them with the relevant sentence structure [14]. The library was built by compiling security attributes derived from parsing and keyword matching.

However, we could not find a knowledge resource that combines all the concepts, relations, and best practices needed during the requirements specification.

To fill the gaps identified in the state of the art, this paper proposes a taxonomy with the concepts and relationships of existing templates and guidelines for requirements specification. The taxonomy will give the requirements and constraints to create a new template.

The paper is organized as follows: The guidelines and templates used to form the taxonomy are presented and detailed in Section II. Section III briefly discusses the analyzed templates. Section IV presents a taxonomy of requirements specification templates. Finally, Section V concludes this work.

II. Baseline Guidelines and Templates

This section lists and explains the different templates and guidelines found in the literature for requirements specification. For a more precise illustration, we have used the example of the Blood Infusion Pump System presented by Lindvall *et al.* [15] to re-specify a requirement of the system using the guidelines and templates, where applicable.

A. Attempto Controlled English (ACE)

ACE, proposed by Schwitter and Fuchs [4], is a computer-processable subset of English with restricted grammar and domain-specific vocabulary that allows specialists to formulate requirement specifications. ACE is associated with its parser (APE) and a reasoner (RACE). ACE is structured by simple, composite, and interrogative sentences.

- **Simple sentences** express a true state of affairs, and their form is *subject* + *predicator* (+ *complement* + *adjunct*).
- **Composite sentences** are built from simpler sentences and constructors: coordination (*and*,*or*, *either-or*), subordination (*if-then*, *who/which/that*), and negation (*not*).
- **Interrogative sentences** allow users to pose *yes/no* questions and *wh*-questions.

ACE is based on syntactic, semantic, and disambiguation principles.

- **Syntactic principles** illustrate a form of the sentence of requirement. For instance, a syntactic principle is "nouns are always used with a determiner" (*e.g.*, "The customer enters a card").
- **Semantic principles** represent how different parts of the requirement are interpreted. For instance, "verbs denote events and states".
- **Disambiguation principles** are the steps taken to decrease and limit the ambiguity in the requirement. For instance, "some ambiguous constructs are not available" (*e.g.*, "John and Mary enter a card") and "unambiguous alternatives with scope markers can be used instead" (*e.g.*, "John and Mary enter a card each").

B. Gellish

Van Renssen presents an application-independent language that allows textual modeling requirements through standard reference data for system customization, data integration, and data exchange [6]. It is based on ontological concepts, where like in an ontology, the Gellish language represents the relationship between two objects. It adopts a table form to represent the language with the following columns:

- **Left-hand object UID:** a unique ID associated with the source object of the relationship.
- **Left-hand object name:** the name of the relationship's source object.
- **Fact UID:** a unique ID of the relationship (or fact).
- **Relationship type name:** the name of the relationship or fact. For example, *is a specialization of*, *is part of*.
- **Right-hand object UID:** a unique ID associated with the target object of the relationship.
- **Right-hand object name:** the name of the relationship's target object.

"Gellish is not limited to specific application domains, although the current ontology (the dictionary) does not yet cover the scope of a natural language". Table I illustrates a fact specified by Gellish.

TABLE I
Gellish Example.

Left-hand object UID	Left-hand object name	Fact UID	Relationship type name	Right-hand object UID	Right-hand object name
111	Andries	11	is classified as a	990007	man

C. Nayak et al.'s template

Nayak *et al.* propose a reliable requirements specification template [5] (based on Volere template [16]) with some parameters that evaluate the reliability of the individual requirement before finalizing the requirements documentation from the initial phase of software development. The template contains information about a requirement, such as Rationale, Source, Requirement Type, etc., but it focuses on the following reliability parameters: Severity Level, Confidence Level, and Rank of Requirement. However, this template does not give a structure for the requirement itself and considers it a description ("a one sentence statement of intention of the requirement") written entirely in natural language.

D. Rupp et. al.'s template

Rupp *et. al.* propose a generic syntactic requirements specification template, which focuses on the syntax (structure) of a requirement, not its semantics (content) [7]. The following parts compose the template.

- **Condition:** condition or constraint under which the requirement is to take place. *<When? Under what conditions?>*.
- **System:** name of the system concerned by the requirement. *THE SYSTEM <system name>*.
- **Priority:** degree of the legal obligation of the requirement. *SHALL, SHOULD, WILL, MAY*.
- **Functional activity:** functionality to be provided by the system:
  - Independent system action: *<process verb>*
  - User interaction: *PROVIDE <whom?> WITH THE ABILITY TO <process verb>*
  - Interface requirement: *BE ABLE TO <process verb>*
- **Object:** object concerned by the requirement. *<object>*.
- **Additional object details:** additional details or explanation about the object of the requirement, for example, *where?* or *how? <additional details about the object>*.

E. Easy Approach to Requirements Syntax (EARS)

Mavin *et al.* present a set of structural rules to address common requirements problems such as ambiguity, complexity, and vagueness [3]. It is based on ECA, "In ECA, the event specifies the signal that triggers the rule and the condition is a logical test that (if satisfied) causes the specified system action" [3]. The rule set allows all requirements to be expressed in natural language in one of the following simple templates.

- **Generic requirements syntax:** *<optional preconditions> <optional trigger> the <system name> shall <system response>*.
- **Ubiquitous requirements - no pre-condition:** *The <system name> shall <system response>*.
- **Event-driven requirements - triggering event:** *WHEN <optional preconditions> <trigger> the <system name> shall <system response>*.
- **Unwanted behaviors:** *IF <optional preconditions><trigger>, THEN the <system name> shall <system response>*.
- **State-driven requirements:** *WHILE <in a specific state> the <system name> shall <system response>*.
- **Optional features:** *WHERE <feature is included> the <system name> shall <system response>*.

F. Mazo et al.'s template

Mazo *et al.* identify some gaps in Rupp *et al.*'s template and, based on those gaps, propose a more robust template that facilitates the work of the requirements engineers and, therefore, improves the quality of the products specified with the new template [8]. Mazo *et al.*'s template is adapted to product lines and auto-adaptive systems (using the RELAX language presented in [17]). It has the following sections.

- **Conditions under which a behavior occurs:** describe behaviors performed or provided only under specific conditions (*e.g., IF, WHILE, IN CASE, AFTER*).
- **Family of systems, systems, or parts of a system:** allows specifying the name of the product line, system, subsystem, or system component (*e.g., ALL SYSTEMS OF THE <product line name>*).
- **Degree of priority:** specifies the degree of priority associated with a requirement (*i.e. SHALL, SHOULD, COULD, WILL*).
- **Activity:** specifies the characterization of the activity conducted by the system or the systems of a product line.
- **Object or objects:** specifies the object or objects that make up the system (*e.g., EACH <object>*).
- **Complementary details of the object(s):** can be one or more adjectives or a more enhanced description of the object.
- **Conditionality in the object:** describes a behavior that the system must execute if and only if the object attains the specified condition (*i.e., IF AND ONLY IF <condition>*.
- **Verification criterion (adjustment) of the requirement:** a detectable criterion to determine to what degree the requirement is verifiable.
- **Relax requirements statements for self-adaptive systems:** represents the autonomous nature of requirements in self-adaptive systems (*e.g., AS MANY, UNTIL*).

G. Cube

Pabuccu *et al.* present the Cube requirement writing template dedicated to software systems and based on 5W1H questions [9].

- **Why:** the goal of the requirement.
- **Who:** the actor of the requirement.
- **Where:** the place of the requirement.
- **When-trigger:** pre-condition of the requirement.
- **When-condition:** post-condition of the requirement.
- **What:** the requirement's action or activity.
- **How:** it explains how the system will be developed.

Cube identifies three types of requirements and provides three different templates:

- **Business Requirement:** *WHO - WHAT- WHERE - WHEN TRIGGER- WHEN CONDITION Aim, Reason: WHY*.
- **User Requirement:** *- when WHO - WHEN TRIGGER - WHERE - What - if - WHEN CONDITION - HOW (optional) - WHY (optional)*.
- **Functional Non-functional Requirement:** *Who - What - when - When Trigger -and When Condition - Where - WHY (optional) - HOW (optional)*.

H. Esser and Struss's template

Esser and Strauss propose a natural language template-based interface to acquire requirements for functional testing of control software for passenger vehicles [10]. The content of the filled-in templates can be represented in propositional logic and temporal relationships and form the correct expected behavior model. The template is structured as an ***if (start-condition) - then (consequence) - until (end-condition)*** sentence (*e.g.*, "if the system is in mode m1, lamp L3 is off, and button B4 is released, then immediately lamp L3 is lit until button B4 is down again or the system leaves m1").

I. ReSA

Mahmud *et al.* propose a toolchain for structured requirements specification in the ReSA language [11]. "ReSA is an ontology-based requirements specification language tailored to automotive embedded systems development". ReSA "(i) renders natural language terms (words, phrases), and syntax, (ii) uses an ontology that defines concepts and syntactic rules of the specification, and (iii) uses requirements boiler-plates to structure specification". It is composed of six boilerplates or templates.

- **Simple:** statement that contains a modal verb, such as, *shall (e.g.*, "system shall be activated").
- **Proposition:** proposition or assertive statement *(e.g.*, "button is pressed".
- **Complex:** "is constructed from a Simple, Proposition boilerplate, and an adverbial conjunctive (such as *while, when, until*)", *e.g.*, "the error shall be reported while the fault is present".
- **Compound:** "is composed of two or more Simple or Proposition boilerplates and the logical operators, *AND/OR*" (*e.g.*, "system shall be activated and driver shall be notified").
- **Conditional:** "a different variant of conditional statements, *i.e.*, *if, if-else, if-elseif, or if-elseif-else*, and conditional nesting".
- **Prepositional Phrase:** "can be used to describe timing properties, the occurrence of events, and other complements to the subject of a main phrase" (*e.g.*, "within 5ms, by the driver".

J. SImple REuse of software requiremeNts (SIREN)

Toval *et al.* present a method for eliciting and specifying system and software requirements. The method includes a repository with reusable requirements, a spiral process model, and a set of requirements document templates [12]. The method focuses on information systems security. It does not provide a structured template for the requirements description but for the entire Software Requirements Specification document. The repository contains two types of requirements: **parameterised** (some parts have to be adapted to the application being developed at the time, *e.g.*, "The security manager shall check the user's identifiers every [time in months] to detect which ones have not been used in the last [time in months]") and **non-parameterised** (could be applied directly to any project concerning the repository's profiles and/or domains, *e.g.*, "The firewall configuration will be screened host").

K. Firesmith template

Firesmith discusses the value of reusable parameterized templates for specifying security requirements [13]. Firesmith outlined an asset-based risk-driven analysis approach for determining the appropriate current parameters to use when reusing parameterized templates to specify security requirements that should improve the quality of security requirements in requirements specifications. Firesmith explains how to create templates and gives an example of a security requirement to specify integrity: "The [application/component/data center/business unit] shall protect the [identifier—type] data it transmits from corruption (*e.g.*, unauthorized addition, modification, deletion, or replay) due to [unsophisticated/somewhat sophisticated/sophisticated] attack during execution of [a set of interactions/use cases] as indicated in [specified table]". However, no general template was provided.

L. Kamalrudin et al.'s template

Kamalrudin *et al.* propose a security requirements library and template to assist the requirements engineer in writing security requirements by providing them with the sentence structure [14]. The library was built by compiling security attributes derived from parsing and keyword matching. The template provided is as follows: *The <Subject> should <Verb/Security Keyword> to the <Object> <Security Keyword> <Security Mechanism> in order to <Adjective Phrase>* (*e.g.*, "[The] customer should register [to] the system using unique username and password in order to proceed to book ticket service").

M. AMAN-DA

Souag *et al.* propose a security requirements elicitation and an analysis method [18]. It uses a multi-level domain ontology of security notions and provides a security requirements template: *<When><Under what condition> <Agent name> "Shall/Should/Will" <Action> <Assets> <Additional Information>* (*e.g.*, "The web publishing system should lock accounts after reaching logon failure threshold").

III. Application Example

Table II shows the re-specification of some Blood Infusion Pump System requirements using templates discussed in Section II. Per the result of the requirements re-specification and the information provided in Section II concerning each guideline and template, we were able to categorize them according to their structure or nature, the type of requirements they can represent, and their output. We also noticed a similarity in the structure of some templates. It is because they were based on a combination of other ones. For example, Mazo *et al.*'s template uses the structure of Rupp *et al.*'s and the EARS templates. This categorization resulted in the creation of the taxonomy presented in Section IV. For example, it is explained by Mavin *et al.*, and clear in the structure of EARS, that the EARS template integrates the event-condition-action concepts of the Event-Condition-Action (ECA) language or template. This is also true in the Mazo *et al.* template, which inherits some concepts, quality attributes, and good practices from the Rupp *et al.* and EARS templates.

IV. Taxonomy

To identify the source and the guidelines and templates used in constructing each template, we propose a taxonomy of all

TABLE II
EXAMPLE REQUIREMENTS USING THE TEMPLATES.

Template	Example Requirement - Blood Infusion Pump System
ACE	**R1.2** If the silent warning alarm persists for 10 seconds or more, then the system shall trigger an audible warning alarm
	R1.2.ACE If the silent warning alarm persists for 10 seconds or more, then the system triggers an audible warning alarm
Rupp *et al.*	**R2.5:** If BP improves from true critical condition to true warning condition, then the system shall trigger a true warning alarm
	R2.5.Rupp: <If BP improves from true critical condition to true warning condition then>$_{condition}$<the blood infusion pump system>$_{system}$ <shall>$_{priority}$ <trigger>$_{activity}$ <an audible critical alarm>$_{object}$
EARS	**R2.8:** The system shall disable the “Start infusion” button whenever blood infusion is stopped due to critical alarm
	R2.8.EARS: The <blood infusion pump system>$_{systemname}$ shall <disable the “Start infusion” button whenever blood infusion is stopped due to critical alarm>$_{systemresponse}$
Mazo *et al.*	**R3.1:** If the system received two consecutive out-of-range BP low values (less than 10) or high values (greater than 180), then the system shall initiate system shutdown
	R3.1.Mazo: If <the system received two consecutive out-of-range BP low values (less than 10) or high values (greater than 180)>$_{condition}$ then <the blood infusion pump system>$_{system}$ <shall>$_{priority}$ <initiate>$_{activity}$ <system shutdown>$_{object}$
Cube	**R1.6:** The system shall disable the ‘Start Infusion’ button during the warning alarm
	R1.6.Cube: <The system>$_{who}$ <shall disable the ‘Start Infusion’ button>$_{what}$ <when the warning alarm starts>$_{whentrigger}$<and enable the button after the alarm stops>$_{whencondition}$
Esser and Struss	**R2.2:** If the silent critical alarm clears within 10 seconds and there was a true warning alarm before, then the system shall resume blood infusion and clear the warning alarm
	R2.2.Esser: If -the silent critical alarm clears within 10 seconds and there was a true warning alarm before-, then -the system shall resume blood infusion and clear the warning alarm- until ?
ReSA	**R1.3:**If BP improves, then the system shall resume blood infusion and clear the warning alarm
	R1.3.ReSA: <If BP improves,>$_{conditional}$ <the system shall resume blood infusion and clear the warning alarm>$_{compound}$
Kamalrudin *et al.*	**S2.2:** The system shall use the authenticators to validate the source of the BP value
	S2.2.Kamal: <The system>$_{subject}$ <shall authenticate>$_{verb}$ <the BP value>$_{object}$ <using>$_{Security-keyword}$ <an authenticator>$_{security-mechanism}$ <to validate the source of the BP value>$_{adjectivephrase}$
AMAN-DA	**S4.1.1:** A known-good cryptographic algorithm shall be used to implement authentication.
	S4.1.1.AMAN-DA: <The system>$_{agent}$ <shall><implement >$_{action}$ < authentication>$_{asset}$ <using a known-good cryptographic algorithm >$_{additionalinformation}$

the above templates in Figure 1. A relationship between two templates means that the target (target of the arrow) template has been considered or studied when constructing the source template (source of the arrow). The benefit of having such a taxonomy is to avoid repeating already included templates when building new requirement templates. The taxonomy categorizes the templates and guidelines between controlled-natural languages and templates. Then, each category determines the type of requirements that can be represented by each template (functional, non-functional, or security requirements). There are also two types of templates identified in the taxonomy, templates for requirements description and templates for System Requirements Specification (SRS) documents, as shown in the legend of Figure 1. For example, as we have mentioned in Section III, Mazo *et al.*'s template integrates some aspects of the Rupp *et al.* and EARS templates. It is evident through the example given in Table II that the syntax of the requirement formulated with the Mazo *et al.*'s template is similar to the requirement reformulated with the Rupp *et al.*'s template. Both requirements are composed of a condition, a system name, a priority keyword, an action, and an object. This similarity is described in Figure 1 by the relationship between Mazo *et al.*'s template (source) and the EARS and Rupp *et al.* templates (targets). This relationship indicates that Mazo *et al.*'s template already respects and integrates the concepts, quality attributes, and good practices from Rupp *et al.* and EARS. Thus, it is unnecessary to reconsider them when using the Mazo*et al.*'s template, which reduces the time and resource cost.

V. CONCLUSION

System security has become a very critical issue to handle at all stages of system design, starting from the requirements specification stage. To guide and simplify this stage, several requirements templates have been proposed. We have discussed the concepts, attributes, and good practices and examined these templates in Section II. Then, in Section III, we re-specified some requirements of the Blood Infusion Pump System using the templates found. We could categorize the templates and guidelines according to different characteristics in this process. However, most of these templates were built based on other templates, where they integrate the concepts, quality attributes, and good practices of others. With the categorization of these templates and the identification of the sources of each template, we were able to create a taxonomy. The taxonomy presented in Section IV aims to avoid the repeated study of source templates which is a loss of research hours and resources. We intend to use this taxonomy as a baseline to create a new security requirements specification template where we can identify, using the taxonomy, the templates we can base our new template on while avoiding re-studying already integrated templates.

ACKNOWLEDGMENT

We thank the French DGA (Direction Générale de l'Armement), the European Union, and the Spanish Government for supporting this research. This work was supported by the project *Data-pl* funded by FEDER/Ministry of Science and Innovation — State Research Agency; the *COPERNICA*

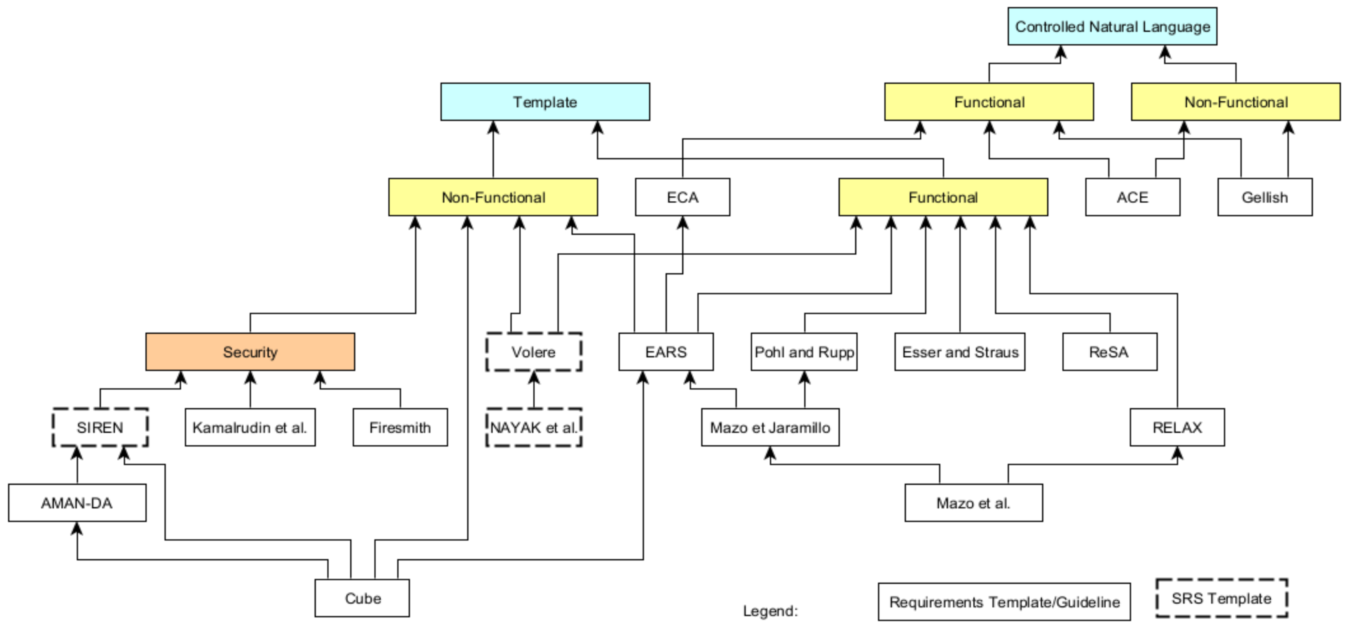


Fig. 1. Requirements templates and guidelines taxonomy.

(P20_01224) and *METAMORFOSIS* (FEDER_US-1381375) projects funded by Junta de Andalucía.

REFERENCES

[1] C. Denger, D. M. Berry, and E. Kamsties, “Higher quality requirements specifications through natural language patterns,” *Proceedings 2003 Symposium on Security and Privacy*, pp. 80–90, 2003.
[2] F. Dalpiaz, I. Schalk, and G. Lucassen, *Pinpointing Ambiguity and Incompleteness in Requirements Engineering via Information Visualization and NLP*, 03 2018, pp. 119–135.
[3] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, “Easy approach to requirements syntax (ears),” in *Proceedings of Requirements Engineering Conference, 2009. RE ’09. 17th IEEE International*. United States: IEEE, Nov. 2009, pp. 317–322, 2009 17th IEEE International Requirements Engineering Conference ; Conference date: 31-08-2009 Through 04-09-2009.
[4] N. Fuchs and R. Schwitter, “Attempto controlled english (ace),” *CoRR*, vol. cmp-lg/9603003, 03 1996.
[5] S. K. Nayak, R. A. Khan, and M. R. Beg, “A comparative template for reliable requirement specification,” *International Journal of Computer Applications*, vol. 14, no. 2, pp. 27–30, 2011.
[6] A. van Renssen, “Gellish: an information representation language, knowledge base and ontology,” *ESSDERC 2003. Proceedings of the 33rd European Solid-State Device Research - ESSDERC ’03 (IEEE Cat. No. 03EX704)*, pp. 215–228, 2003.
[7] C. Rupp, M. Simon, and F. Hocker, “Requirements engineering und management,” *HMD Praxis der Wirtschaftsinformatik*, vol. 46, pp. 94–103, 06 2014.
[8] R. Mazo, C. M. Z. Jaramillo, P. Vallejo, and J. M. Medina, “Towards a new template for the specification of requirements in semi-structured natural language,” *J. Softw. Eng. Res. Dev.*, vol. 8, p. 3, 2020.
[9] Y. U. Pabuccu, I. Yel, A. B. Helvacioglu, and B. N. Asa, “The requirement cube: A requirement template for business, user, and functional requirements with 5w1h approach,” *International Journal of Information System Modeling and Design (IJISMD)*, vol. 13, no. 1, pp. 1–18, 2022.
[10] M. Esser and P. Struss, “Obtaining models for test generation from natural-language-like functional specifications,” pp. 75–82, 2007.
[11] N. Mahmud, C. Seceleanu, and O. Ljungkrantz, “Resa tool: Structured requirements specification and sat-based consistency-checking,” in *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2016, pp. 1737–1746.
[12] A. Toval, J. Nicolás, B. Moros, and F. García, “Requirements reuse for improving information systems security: a practitioner’s approach,” *Requirements Engineering*, vol. 6, no. 4, pp. 205–219, 2002.
[13] D. Firesmith, “Specifying reusable security requirements.” *J. Object Technol.*, vol. 3, no. 1, pp. 61–75, 2004.
[14] M. Kamalrudin, N. Mustafa, and S. Sidek, “A template for writing security requirements,” in *Asia Pacific Requirements Engeneering Conference*. Springer, 2017, pp. 73–86.
[15] M. Lindvall, M. Diep, M. Klein, P. Jones, Y. Zhang, and E. Vasserman, “Safety-focused security requirements elicitation for medical device software,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 134–143.
[16] J. Robertson and S. Robertson, “Volere,” *Requirements Specification Templates*, 2000.
[17] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, “Relax: Incorporating uncertainty into the specification of self-adaptive systems,” in *2009 17th IEEE International Requirements Engineering Conference*, 2009, pp. 79–88.
[18] A. Souag, R. Mazo, C. Salinesi, and I. Comyn-Wattiau, “Using the aman-da method to generate security requirements: a case study in the maritime domain,” *Requirements Engineering*, vol. 23, no. 4, pp. 557–580, 2018.

Estimating Functional Size of Software with Confidence Intervals

Luigi Lavazza Angela Locoro
Dipartimento di Scienze Teoriche e Applicate
Università degli Studi dell'Insubria
Varese, Italy
email:luigi.lavazza, angela.locoro@uninsubria.it

Roberto Meli
DPO
Rome, Italy
email:roberto.meli@dpo.it

***Abstract*—In many projects, software functional size is measured via the IFPUG (International Function Point Users Group) Function Point Analysis method. However, applying Function Point Analysis using the IFPUG process is possible only when functional user requirements are known completely and in detail. To solve this problem, several early estimation methods have been proposed and have become *de facto* standard processes. Among these, a prominent one is the 'NESMA (Netherlands Software Metrics Association) estimated' (also known as High-level Function Point Analysis) method. The NESMA estimated method simplifies the measurement by assigning fixed weights to Base Functional Components, instead of determining the weights via the detailed analysis of data and transactions. This makes the process faster and cheaper, and applicable when some details concerning data and transactions are not yet known. The accuracy of the mentioned method has been evaluated, also via large-scale empirical studies, showing that the yielded approximate measures are sufficiently accurate for practical usage. However, a limitation of the method is that it provides a specific size estimate, while other methods can provide confidence intervals, i.e., they indicate with a given confidence level that the size to be estimated is in a range. In this paper, we aim to enhance the NESMA estimated method with the possibility of computing a confidence interval. To this end, we carry out an empirical study, using data from real-life projects. The proposed approach appears effective. We expect that the possibility to estimate that the size of an application is in a range will help project managers deal with the risks connected with inevitable estimation errors.**

***Index Terms*—Function Point Analysis; Early Size Estimation; High-Level FPA; NESMA estimated.**

I. Introduction

In the late seventies, Allan Albrecht introduced Function Points Analysis (FPA) at IBM [1], as a means to measure the functional size of software, with special reference to the "functional content" delivered by software providers. Albrecht aimed at defining a measure that might be correlated to the value of software from the perspective of a user, and could also be useful to assess the cost of developing software applications, based on functional user requirements.

FPA is a Functional Size Measurement Method (FSMM), compliant with the ISO/IEC 14143 standard, for measuring the size of a software application in the early stages of a project, generally before actual development starts. Accordingly, software size measures expressed in Function Points (FP) are often used for cost estimation.

The International Function Points User Group (IFPUG) is an association that keeps FPA up to date, publishes the official FP counting manual [2], and certifies professional FP counters. Unfortunately, in some conditions, performing the standard IFPUG measurement process may be too long and expensive, with respect to management needs, because standard FP measurement can be performed only when relatively complete and detailed requirements specifications are available, while functional measures could be needed much earlier for management purposes.

To tackle this problem, the IFPUG proposes Simple Function Points (SFP). This is an alternative way of measuring the functional size of software: while the SFP method is based on the same concepts as FPA, it requires less detailed information than FPA, so that it is applicable before complete and detailed requirements specifications are available; besides, if is faster and cheaper to apply. As such, it is often presented as a lightweight functional measurement method, also suitable for agile processes. Although the SFP method provides measures that are quantitatively similar to those yielded by FPA, it is not an approximation method for FPA; instead, it is a different measurement method that yields different measures.

Before SFP was proposed, many methods were invented and used to provide *estimates* of functional size measures, based on fewer or coarser-grained information than required by standard FPA. These methods are applied very early in software projects, even before deciding what process (e.g., agile or waterfall) will be used. Among these methods, one of the most widely used is the "NESMA estimated" method [3], which was developed by NESMA [4]. Using this method for size estimation was then suggested by IFPUG [5], which renamed the method High-Level FPA (HLFPA).

HLFPA has been evaluated by several studies, which found that the method is usable in practice to approximate traditional FPA values, since it yields reasonably accurate estimates, although it has been observed that the NESMA method tends to underestimate size, which is potentially dangerous.

Many estimation methods provide a "confidence interval", meaning that instead of providing a single value, they predict that the size is in an interval. The greater the required confidence, the greater the interval. Knowing the confidence interval is considered very useful by project managers, because it helps managing the risk deriving from inevitable estimation errors and the inherent uncertainty of estimates. Unfortunately, the NESMA estimated method does not provide a confidence interval. This papers aim to enhancing the NESMA estimated method by equipping it with a mechanism to create a confidence interval.

The remainder of the paper is organized as follows. Section II provides an overview of FPA and the NESMA method. Section III describes the empirical study and its results, which are discussed in Section IV. In Section V, we discuss the threats to the validity of the study. Section VI reports about related work. Finally, in Section VII, we draw some conclusions and outline future work.

II. Background

Function Point Analysis was originally introduced by Albrecht to measure the size of data-processing systems from the point of view of end-users, with the goal of the estimating the value of an application and the development effort [1]. The critical fortunes of this measure led to the creation of the IFPUG (International Function Points User Group), which maintains the method and certifies professional measurers.

The "amount of functionality" released to the user can be evaluated by taking into account 1) the data used by the application to provide the required functions, and 2) the transactions (i.e., operations that involve data crossing the boundaries of the application) through which the functionality is delivered to the user. Both data and transactions are counted on the basis of Functional User Requirements (FURs) specifications, and constitute the IFPUG Function Points measure.

FURs are modeled as a set of Base Functional Components (BFCs), which are the measurable elements of FURs: each of the identified BFCs is measured, and the size of the application is obtained as the sum of the sizes of BFCs. IFPUG BFCs are: data functions (also known as logical files), which are classified into Internal Logical Files (ILF) and External Interface Files (EIF); and Elementary Processes (EP)—also known as transaction functions—which are classified into External Inputs (EI), External Outputs (EO), and External inQuiries (EQ), according to the activities carried out within the considered process and the primary intent.

The complexity of a data function (ILF or EIF) depends on the RETs (Record Element Types), which indicate how many types of variations (e.g., sub-classes, in object-oriented terms) exist per logical data file, and DETs (Data Element Types), which indicate how many types of elementary information (e.g., attributes, in object-oriented terms) are contained in the given logical data file.

The complexity of a transaction depends on the number of FTRs—i.e., the number of File Types Referenced while performing the required operation—and the number of DETs—i.e., the number of types of elementary data—that the considered transaction sends and receives across the boundaries of the application. Details concerning the determination of complexity can be found in the official documentation [2].

The core of FPA involves three main activities:

1) Identifying data and transaction functions.
2) Classifying data functions as ILF or EIF and transactions as EI, EO or EQ.
3) Determining the complexity of each data or transaction function.

The first two of these activities can be carried out even if the FURs have not yet been fully detailed. On the contrary, activity 3 requires that all details are available, so that FP measurers can determine the number of RET or FTR and DET involved in every function. Activity 3 is relatively time- and effort-consuming [6].

HLFPA does not require activity 3, thus allowing for size estimation when FURs are not fully detailed: it only requires that the complete sets of data and transaction functions are identified and classified.

The SFP method [7] does not require activities 2 and 3: it only requires that the complete sets of data and transaction functions are identified.

Both the HLFPA and SFP methods let measurers skip the most time- and effort-consuming activity, thus both are relatively fast and cheap. The SFP method does not even require classification, making size estimation even faster and less subjective (since different measurers can sometimes classify differently the same transaction, based on the subjective perception of the transaction's primary intent).

NESMA defined two size estimation methods: the 'NESMA Indicative' and the 'NESMA Estimated' methods. IFPUG acknowledged these methods as early function point analysis methods, under the names of 'Indicative FPA' and 'High-Level FPA,' respectively [5]. The NESMA Indicative method proved definitely less accurate [8], [9]. Hence, in this paper, we consider only the NESMA Estimated method.

The NESMA Estimated method requires the identification and classification of all data and transaction functions, but does not require the assessment of the complexity of functions: ILF and EIF are assumed to be of low complexity, while EI, EQ and EO are assumed to be of average complexity. Hence, estimated size is computed as follows:

$$EstSize_{UFP} = 7\ \#ILF + 5\ \#EIF + 4\ \#EI + 5\ \#EO + 4\ \#EQ$$

where *#ILF* is the number of data functions of type ILF, *#EI* is the number of transaction functions of type EI, etc.

III. Empirical study

In this section, the empirical study is described: Section III-A described the dataset used for the reported analysis; Section III-B illustrates some considerations concerning the accuracy of the NESMA method that affect the study; Section III-C describes how the study was performed; finally, SectionIII-D describes the obtained results.

A. The dataset

In the empirical study, we use an ISBSG dataset [10], which has been extensively used for studies concerning functional size [11]–[16].

The ISBSG dataset contains several small project data. As a matter of fact, estimating the size of small projects is not very interesting. Based on these considerations, we removed from the dataset the projects smaller than 100 UFP (Unadjusted Function Points). The resulting dataset includes data from 140 projects having size in the [103, 4202] range. Some descriptive statistics for this dataset are given in Table I.

TABLE I
DESCRIPTIVE STATISTICS FOR THE ISBSG DATASET.

	UFP	*#ILF*	*#EIF*	*#EI*	*#EO*	*#EQ*	NESMA
Mean	801	22	20	35	37	37	730
Std	818	21	22	37	65	48	721
Median	475.5	14	14.5	22	10	20.5	463
Min	103	0	0	0	0	0	71
Max	4202	100	172	204	442	366	3755

B. *The accuracy of the NESMA estimated method*

As already observed in previous papers [16], [17], the NESMA estimated method tends to underestimate. Figure 1 shows that more than 75% of the estimates by NESMA have positive error. Being the error defined as the actual size (i.e., the size measured via the ISBSG standard FPA process) minus the estimate, positive error indicate underestimation.

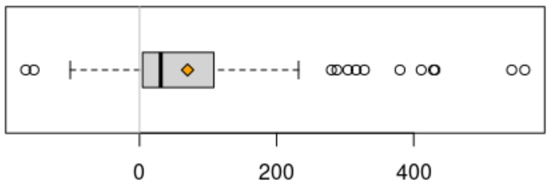


Fig. 1. Histogram of estimation errors by the NESMA method, when applied to the ISBSG dataset.

In addition, Figure 1 suggests that the distribution of NESMA errors is skewed. The skewedness of NESMA errors is clearly visible in Figure 2, which illustrates the distribution of errors: it is easy to notice that most errors are positive.

For our purposes, the fact that the distribution of NESMA errors is skewed and not centered on zero means that we cannot evaluate confidence errors as is usually done. Specifically, given a confidence level C we cannot select two error levels e_L and e_H that are symmetric with respect to the mean error $\bar{e}$ (i.e., $|e_H - \bar{e}| = |\bar{e} - e_L|$) such that the proportion of errors such that $e_H \geq error \geq e_L$ is C.

Since it makes hardly sense to provide confidence intervals for a method that underestimates systematically, we first "correct" the NESMA estimated method. Via a trial-and-error procedure, we found that by multiplying NESMA estimates by 1.08 it is possible to obtain estimates that have a better error distribution (less skewed and centered around zero) and a smaller mean absolute error (50.7 UFP instead of 83.8 UFP).

The boxplot of estimation errors obtained with the corrected NESMA method is shown in Figure 3. The error distribution is shown in Figure 4: it can be noticed that the distribution is much less skewed than in Figure 2.

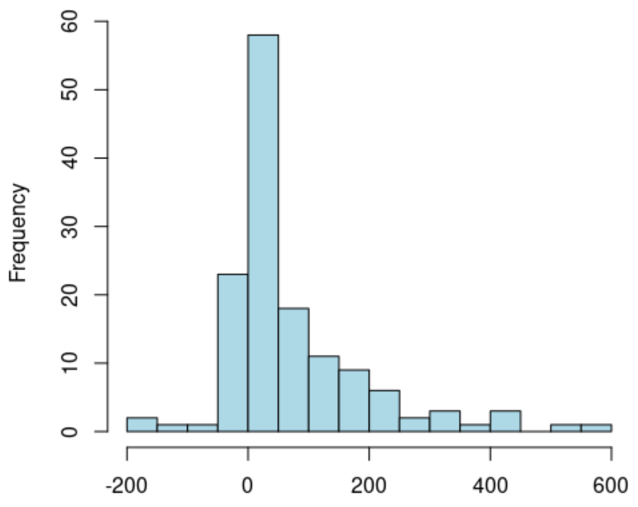


Fig. 2. Histogram of estimation errors by the NESMA method, when applied to the ISBSG dataset.

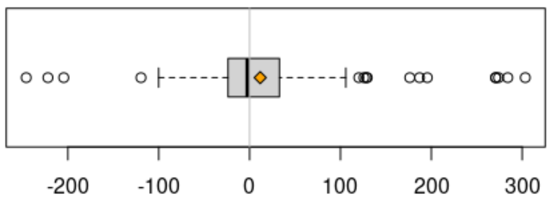


Fig. 3. Boxplot of estimation errors by the corrected NESMA method, when applied to the ISBSG dataset.

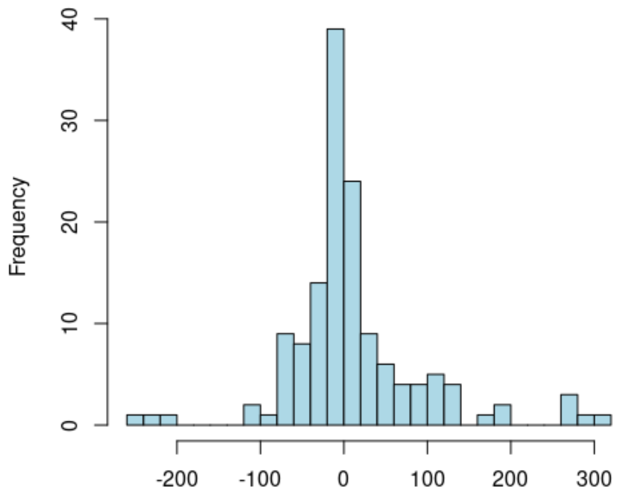


Fig. 4. Boxplot of estimation errors by the corrected NESMA method, when applied to the ISBSG dataset.

Since the practical objective of this work is to provide project managers with reliable predictions of functional size, in what follows we consider only estimates provided by the original NESMA method and corrected as described above. In other words, we consider the following estimates:

$$\mathit{EstSize}_{UFP} = 1.08\ (7\ \mathit{\#ILF}+5\ \mathit{\#EIF}+4\ \mathit{\#EI}+5\ \mathit{\#EO}+4\ \mathit{\#EQ})$$

We make reference to this estimation as the "Corrected NESMA" method.

C. Method used

In essence, given a confidence level C we aim at finding two values k_L and k_H such that a proportion C of the actual size measures (i.e., measures obtained via the official IFPUG FPA process) is in the range $[k_L \cdot \mathit{EstSize}_{UFP}, k_H \cdot \mathit{EstSize}_{UFP}]$, where $\mathit{EstSize}_{UFP}$ is the size estimates computed via the Corrected NESMA method.

Finding k_L and k_H would be straightforward if the estimation errors obtained via the Corrected NESMA method were normally distributed. Instead, it is not so, as shown by the Shapiro-Wilk test.

Therefore, we proceeded as follows:

1) We computed the ratio $\frac{\mathit{ActualSize}}{\mathit{EstSize}_{UFP}}$ for all projects in the dataset, obtaining a set of ratios; this set was then sorted and stored in vector *vRatios*.
2) We computed the quantiles from 0 to 1, with 0.01 steps, of *vRatios*, obtaining an ordered vector *vQuant*.
3) We looked for two indexes i_L and i_H in *vQuant* such that $i_H - i_L + 1 = C \cdot n$ (where *n* is the number of projects in the dataset).
4) k_L and k_H are the values in *vRatios* having index i_L and i_H, respectively, i.e., $vRatios[i_L]$ and $vRatios[i_H]$.

In this way, we obtain a size estimate interval that contains a proportion C of all estimates, such that all estimates outside the interval are greater than those within the interval.

D. Results obtained

We applied the procedure described in Section III-C for various confidence levels. The results obtained are given in Table II. Note that these results depend on the dataset being used, in our case, the ISBSG dataset. In other contexts, a given confidence level could correspond to different confidence intervals. For instance, in the ISBSG dataset, the minimum and maximum ratios $\frac{\mathit{ActualSize}}{\mathit{EstSize}_{UFP}}$ are 0.758 and 1.343, respectively; in another dataset, a smaller minimum and a larger maximum ratios are clearly possible.

TABLE II
CONFIDENCE INTERVALS FOR VARIOUS CONFIDENCE LEVELS.

conf. level	k_L	k_H
0.50	0.947	1.053
0.60	0.933	1.069
0.70	0.902	1.100
0.80	0.875	1.134
0.90	0.840	1.165
0.95	0.826	1.220
1.00	0.758	1.343

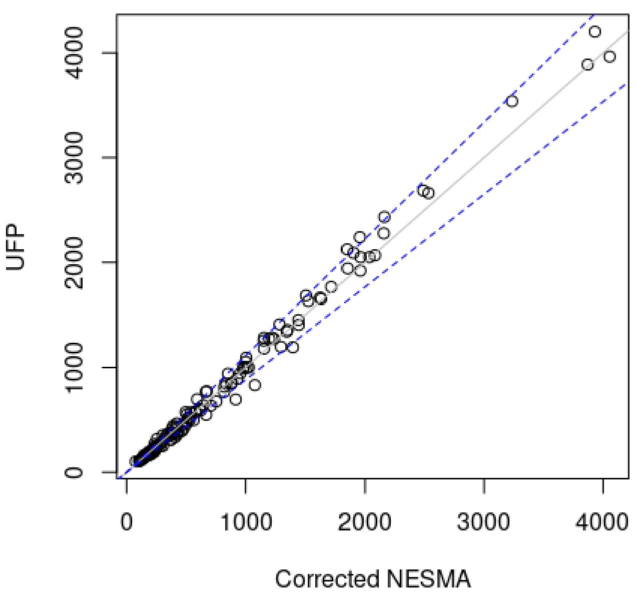


Fig. 5. Corrected NESMA estimates vs. actual size in UFP, with confidence $C = 0.75$.

For illustration purposes, Figure 5 plots the ISBSG project data in the plan defined by actual size (the y axis) and the size estimated via the Corrected NESMA method (the x axis). In the plot, the dashed blue lines represent the $y = k_L\ x$ and $y = k_H\ x$ lines.

IV. DISCUSSION OF RESULTS

In the previous sections, we exploited a dataset that collects measures from real-life projects to determine a) a correction of the estimates provides by the NESMA method, and b) confidence intervals for the corrected estimates.

The contribution of this paper is twofold:

– Organizations that own historical data like those we used can apply the procedure illustrated in Sections III-B and III-C to derive the correction constant and the confidence intervals that suite best their development process.
– Organizations that do not own historical data like those we used can use our correction constant (1.08) and the intervals in Table II, to get an idea of how much estimates obtained via the NESMA method can vary in practice. However, these organizations should be aware that the data we used might not match their situations, hence both the correction constant and the confidence intervals might not be perfectly suited for their case.

The confidence interval can be used to perform risk analysis. For instance, Table II shows that, given an estimate already corrected with respect to the NESMA original prediction, there are 30% probabilities that the actual size is more than 10% different (greater or smaller) than estimated. Most likely, half of these 30% probabilities concern the underestimation case: as a result, a project manager should consider that the probability of underestimating functional size of 10% or more is around 15%. The risk concerning the underestimation of cost can be then computed, if the relationship between size and cost is known. So, the proposed method supports typical project management activities, like controlling the risk of size underestimation, that are not supported by the original NESMA size estimation method.

Finally, being the estimates obtained via the Corrected NESMA method proportional to the estimates obtained via the original NESMA method, the confidence intervals for the Corrected NESMA method can be easily converted into confidence intervals for the original NESMA method.

V. Threats to validity

The proposed approach is mostly empirical. From a theoretical point of view, the adopted practices may not be perfect, but the context itself suggests that this is not very relevant. The definition of the NESMA estimated method itself has no theoretically strong basis: it is simply the hypothesis—not experimentally verified—that on average data have low complexity (in FPA terms) while transactions have mid complexity. So, we looked for reasonable confidence intervals, although these intervals are not statistically linked to confidence levels in a rigorous way.

Another typical concern in this kind of studies is the generalizability of results outside the scope and context of the analyzed dataset. In our case, the ISBSG dataset is deemed the standard benchmark among the community, and it includes data from several application domains. Therefore, our results may be representative of a fairly comprehensive situation. However, additional studies are needed for confirming the general validity of this study. In the meanwhile, readers are reminded that the amount by which the NESMA method underestimates depend on the considered dataset; similarly, the confidence interval depends on the dataset. In both cases, the distribution of the complexity of BFCs (i.e., ILF, EIF, EI, EO and EQ) rules.

VI. Related Work

Measures for early software estimation were conceived since the last decades [18]–[20]. The present study aims to advance this field by providing statistical foundations to some of these measures, by using confidence intervals where approaches not based on probability distributions were adopted. For example, the "Early & Quick Function Point" (EQFP) method [21] estimates an error of $\pm 10\%$ of the real size of software, for most of the times, but fails to indicate a more robust indicator of this estimate, such as a confidence interval. Several other early estimation methods were proposed: Table III lists the most popular ones.

TABLE III
EARLY ESTIMATION METHODS: DEFINITIONS AND EVALUATIONS

Method name	Definition	Used functions	Weight	Evaluation
NESMA indicative	[22] [23]	data	fixed	[3] [17], [24]–[27] [9]
NESMA estimated	[22] [23]	all functions	fixed	[3] [17], [24]–[27] [9]
Early & Quick FP	[20] [28] [21]	all functions	statistics	[9] [29]
simplified FP (sFP)	[30]	all functions	fixed	[9]
ISBSG average weights	[31]	all functions	statistics	[9]
SiFP	[32]	data and trans.	statistics	[11] [13]

Recently, comparisons based on the accuracy of the HLFPA method and statistical modelling methods were carried out in order to assess whether standard measures fail in underestimating or overestimating software size [16].

A survey [33] reports how machine learning techniques were used for software development effort estimation, reporting accuracy as a comparison criterion for all the methods analysed. To the best of our knowledge, confidence intervals are overlooked as robust indicators of the estimates done in software size. In this respect, this study aims to emphasize the importance of providing robust indicators for a more reliable comparison and precision of reporting.

VII. Conclusion

The "NESMA estimated" method was proposed to estimate the functional size of software (expressed in IFPUG Function Points). The NESMA method assigns fixed weights to base functional components (i.e., ILF, EIF, EI, EO and EQ), so that it is not necessary to analyze in depth every logic data file or transaction. This makes the method both easier and faster, and applicable when the details needed to characterize and weight base functional components are not yet available.

Previous studies showed that the NESMA method is sufficiently accurate to be used in practice. However, it has two possibly relevant limitations: 1) it tends to underestimate the "real" (i.e., as obtained via the IFPUG FPA process) size of software, and 2) it yields a single estimate, with no confidence intervals. Both these characteristics can be be problematic for software project managers. In fact, planning a project based on underestimated size and, consequently, on underestimated effort estimates usually leads to unrealistic plans. Besides, getting a confidence interval for size estimates allows for evaluating the risks connected with imprecise size estimates.

In this paper, we have proposed a correction for the estimates yielded by the NESMA method, to avoid underestimation, and a procedure to compute the confidence interval. Both these contributions are expected to make project managers' life easier.

Acknowledgment

The work reported here was partly supported by Fondo per la Ricerca di Ateneo, Università degli Studi dell'Insubria.

References

[1] A. J. Albrecht, "Measuring application development productivity," in Proceedings of the joint SHARE/GUIDE/IBM application development symposium, vol. 10, 1979, pp. 83–92.
[2] International Function Point Users Group (IFPUG), "Function point counting practices manual, release 4.3.1," 2010.
[3] H. van Heeringen, E. van Gorp, and T. Prins, "Functional size measurement-accuracy versus costs–is it really worth it?" in Software Measurement European Forum (SMEF), 2009.
[4] nesma, "nesma site," https://nesma.org/ [retrieved: March, 2023].
[5] A. Timp, "uTip – Early Function Point Analysis and Consistent Cost Estimating," 2015, uTip # 03 – (version # 1.0 2015/07/01).
[6] L. Lavazza, "On the effort required by function point measurement phases," International Journal on Advances in Software, vol. 10, no. 1 & 2, 2017, pp. 108–120.
[7] IFPUG, "Simple Function Point (SFP) Counting Practices Manual Release 2.1," 2021.

[8] nesma, “Early Function Point Analysis,” https://nesma.org/themes/sizing/function-point-analysis/early-function-point-counting/ [retrieved: March, 2023].
[9] L. Lavazza and G. Liu, “An empirical evaluation of simplified function point measurement processes,” Journal on Advances in Software, vol. 6, no. 1& 2, 2013, pp. 1–13.
[10] International Software Benchmarking Standards Group, ““Worldwide Software Development: The Benchmark, release 11,” ISBSG, 2009.
[11] L. Lavazza and R. Meli, “An evaluation of simple function point as a replacement of IFPUG function point,” in IWSM–MENSURA 2014. IEEE, 2014, pp. 196–206.
[12] L. Lavazza, S. Morasca, and D. Tosi, “An empirical study on the effect of programming languages on productivity,” in Proceedings of the 31st Annual ACM Symposium on Applied Computing, 2016, pp. 1434–1439.
[13] F. Ferrucci, C. Gravino, and L. Lavazza, “Simple function points for effort estimation: a further assessment,” in 31st Annual ACM Symposium on Applied Computing. ACM, 2016, pp. 1428–1433.
[14] L. Lavazza, S. Morasca, and D. Tosi, “An empirical study on the factors affecting software development productivity,” E-Informatica Software Engineering Journal, vol. 12, no. 1, 2018, pp. 27–49.
[15] L. Lavazza, G. Liu, and R. Meli, “Productivity of software enhancement projects: an empirical study.” in IWSM-Mensura, 2020, pp. 1–15.
[16] G. Liu and L. Lavazza, “Early and quick function points analysis: Evaluations and proposals,” Journal of Systems and Software, vol. 174, 2021, p. 110888.
[17] L. Lavazza and G. Liu, “An Empirical Evaluation of the Accuracy of NESMA Function Points Estimates,” in ICSEA, 2019, pp. 24–29.
[18] D. B. Bock and R. Klepper, “FP-S: a simplified function point counting method,” Journal of Systems and Software, vol. 18, no. 3, 1992, pp. 245–254.
[19] G. Horgan, S. Khaddaj, and P. Forte, “Construction of an FPA-type metric for early lifecycle estimation,” Information and Software Technology, vol. 40, no. 8, 1998, pp. 409–415.
[20] L. Santillo, M. Conte, and R. Meli, “Early & Quick Function Point: sizing more with less,” in 11th IEEE International Software Metrics Symposium (METRICS’05). IEEE, 2005, pp. 41–41.
[21] DPO, “Early & Quick Function Points Reference Manual - IFPUG version,” DPO, Roma, Italy, Tech. Rep. EQ&FP-IFPUG-31-RM-11-EN-P, April 2012.
[22] NESMA–the Netherlands Software Metrics Association, “Definitions and counting guidelines for the application of function point analysis. NESMA Functional Size Measurement method compliant to ISO/IEC 24570 version 2.1,” 2004.
[23] International Standards Organisation, “ISO/IEC 24570:2005 – Software Engineering – NESMA functional size measurement method version 2.1 – definitions and counting guidelines for the application of Function Point Analysis,” 2005.
[24] F. G. Wilkie, I. R. McChesney, P. Morrow, C. Tuxworth, and N. Lester, “The value of software sizing,” Information and Software Technology, vol. 53, no. 11, 2011, pp. 1236–1249.
[25] J. Popović and D. Bojić, “A comparative evaluation of effort estimation methods in the software life cycle,” Computer Science and Information Systems, vol. 9, no. 1, 2012, pp. 455–484.
[26] P. Morrow, F. G. Wilkie, and I. McChesney, “Function point analysis using nesma: simplifying the sizing without simplifying the size,” Software Quality Journal, vol. 22, no. 4, 2014, pp. 611–660.
[27] S. Di Martino, F. Ferrucci, C. Gravino, and F. Sarro, “Assessing the effectiveness of approximate functional sizing approaches for effort estimation,” Information and Software Technology, vol. 123, July 2020.
[28] T. Iorio, R. Meli, and F. Perna, “Early&quick function points® v3. 0: enhancements for a publicly available method,” in SMEF, 2007, pp. 179–198.
[29] R. Meli, “Early & quick function point method-an empirical validation experiment,” in Int. Conf. on Advances and Trends in Software Engineering, Barcelona, Spain, 2015, pp. 14–22.
[30] L. Bernstein and C. M. Yuhas, Trustworthy systems through quantitative software engineering. John Wiley & Sons, 2005, vol. 1.
[31] R. Meli and L. Santillo, “Function point estimation methods: A comparative overview,” in FESMA, vol. 99. Citeseer, 1999, pp. 6–8.
[32] R. Meli, “Simple function point: a new functional size measurement method fully compliant with IFPUG 4.x,” in Software Measurement European Forum, 2011, pp. 145–152.
[33] M. N. Mahdi, M. H. Mohamed Zabil, A. R. Ahmad, R. Ismail, Y. Yusoff, L. K. Cheng, M. S. B. M. Azmi, H. Natiq, and H. Happala Naidu, “Software project management using machine learning technique—a review,” Applied Sciences, vol. 11, no. 11, 2021, p. 5183.

Projects VS Continuous Product Development – Does it Affect Benefits Realization?

Sinan S. Tanilkan
Center for Effective Digitalization of the Public Sector
Simula Metropolitan
Pb 4, St.Olavs Plass, Oslo, Norway
Email: sinan@simula.no
0000-0003-4216-5172

Jo E. Hannay
Center for Effective Digitalization of the Public Sector
Simula Metropolitan
Pb 4, St.Olavs Plass, Oslo, Norway
Email: johannay@simula.no
0000-0002-8657-7593

***Abstract*—Software investments are traditionally implemented using project organization, which often leads project participants to focus on time, cost and scope, rather than the intended benefits of the investment. We conducted a survey to compare work organized as projects against work organized as Continuous Product Development (CPD). Our results indicate that: 1. Both project organization and CPD are commonly used in practice. 2. Agile is very popular, but DevOps and the use of linear models for organizing work are also frequent. 3. CPD is perceived to outperform projects in realization of benefits. 4. We found no difference in perceived realization of benefits between those using or not using a set of ways of organizing work (including linear models, agile, DevOps, BizDev or program organization). We conclude that organizing work using CPD is a viable alternative to project organization, especially in situations where failure must be avoided. Also, we suggest that more research should be conducted to better understand what factors of the different ways of organizing work affects the realization of benefits.**

Keywords*— *Software Project; Continuous Product Development; Benefits Realization; Agile; DevOps; BizDev.

I. INTRODUCTION

Traditionally, software investments are implemented through projects or programs – sometimes managed using portfolio management. Once a project is finished, the solution is transferred to IT operations, who takes over responsibility for the solution, including its maintenance. This approach fits nicely into financial management, where the temporary organization of projects are considered capital expenditure, and the continuous maintenance done by IT operations are considered operational expenses.

Although there are clearly organizations that are successful in developing software solutions using project organization [1], many studies raise concerns about the low degree of success in software projects [2]–[4].

To better understand project success, Baccarini suggests that *project success = project management success + product success* [5]. In this way of thinking, project management success is concerned with delivering a project according to the agreed time, cost and scope, while product success is concerned with the realization of benefits of investments. When the temporary project organization is dismantled before realization of the benefits of the product begins, project participants tend to prioritize what they can be measured on [6] – which is project management success [7]. Thus, the way software investments are financed is likely to limit the realization of benefits.

An alternative approach to financing software investments, that seems to be popular when talking with practitioners, is the use of Continuous Product Development (CPD). In CPD, the team or organization is tasked to work on a product, or product area, with no defined end date, often as a solid line organization. The cost side of the investment is managed by the amount of people allocated to the product organization, allowing the people involved to focus on the product and the benefits of the product.

There are two important differences between software investments organized as projects vs. CPD. First, although the success criteria should be the same, they often end up being different. While projects have a tendency to focus on project management success, CPD organizations has to deal with the product's success or failure over time. Second, unpredictably is handled differently. While projects have a defined end date within which the agreed scope should be delivered, CPD is often used to deliver product features continuously, handling changes as they come.

As CPD seems a popular way of organizing software work, we wish to explore the degree of adoption of project and CPD. Also, we wish to explore the degree to which one way of organizing work performs better in realizing benefits of the investment. To explore these topics, we conducted a survey among practitioners in the Norwegian IT-industry.

The next section presents work relevant for the research topic, before the research questions are presented in Section III. The research method is presented in Section IV and the results in Section V. After that, we discuss and conclude.

II. BACKGROUND

The organizing of software work has evolved over the years driven by a need to make use of the information and understanding that is gained during the software process, for the purpose of meeting stakeholder needs better. Even the earliest process models had a focus on producing a system that is useful for the customer.

One of the earliest process models for software development was described in 1956 [8], later coined the waterfall model [9]. The waterfall model is a linear model, where requirements are communicated clearly in the beginning of a project, and the project is planned from initiation to completion, providing stability, structure and predictability [10]. In 1970, Royce [11] suggested that understanding gained in one phase, can result in the need to redo work from a previous phase. Royce's take on this problem, was that moving back to previous project phases is costly, and should be avoided by better preparations.

Iterative process models take a different approach to changing understanding. Rather than considering changed understanding as a deviation from the plan, iterative models are designed so that understanding acquired in one iteration of software development, can be utilized in the following iterations. Rather than trying to eliminate the need for adaptation, iterative approaches are designed to handle change proactively, rather than reactively. Although iterative organization of software work has reportedly taken place as early as 1957, the first publications on the topic were only due in 1988 [12].

Although early iterative models helped practitioners to utilize new understanding acquired during development, feedback from users often came late, due to infrequent, perhaps only yearly, releases. From around the year 2000, release cycles started to shorten, and by 2010 companies were releasing software multiple times a day [13]. It is likely that this change has been aided both by agile development and management, and the emergence of approaches such as DevOps.

Agile software development is all about feedback and change [14]. This is aided by principles such as "early and continuous delivery of valuable software", "welcoming changing requirements", and "business people and developers working together daily" [15]. Projects using agile practices, have been found to see themselves as more successful in realizing benefits than other projects [16], especially those with flexible scope and frequent delivery to the client.

DevOps, and later BizDev, were designed to increase information sharing and collaboration among organizational units: "DevOps integrates the two worlds of development and operations, using automated development, deployment, and infrastructure monitoring. It's an organizational shift in which, instead of distributed siloed groups performing functions separately, cross-functional teams work on continuous operational feature deliveries" [17]. This means that those who develop become integrated into where benefit are experienced, which ostensibly, should foster increased understanding of benefits during development. Moreover, just as DevOps brings the organizational units responsible for development and operations closer together, BizDev suggests to bring those making business decisions closer to those developing software solutions, through continuous planning and continuous budgeting. On the face of it, this is perfect for evolving and utilizing understanding of benefits. Combining DevOps and BizDev into BizDevOps [18] would seem better still.

Practices such as agile, DevOps and BizDev fit nicely into CPD organization, because the practices are geared towards continuously learning and delivering value, without the constraints imposed by project organization and more linear approaches. An approach that shares many similarities with CPD is Continuous Software Engineering (CSE) [18]. In CSE it is suggested that software engineering should be considered as a set of continuous processes, including continuous planning, budgeting, integration, delivery, deployment, verification, testing, compliance, security, evolution, use, trust, run-time monitoring, improvement, innovation and experimentation. While all of these processes fit nicely into CPD, they are not a requirement for CPD. What CPD adds to the picture is the explicit lifecycle-focus on product and the discard of the time-bounded project as organizational form.

III. Research Questions

Our first objective is to understand how common different ways of organizing software work are. We look into the adoption of project vs. CPD, and the adoption of ways of organizing, such as the use of linear models, agile, DevOps, BizDev and program organization.

Our second objective is to understand if there are differences in the perceived realization of benefits when using the above ways of organizing.

We pose the following research questions:

RQ1 How common is CPD compared to project organization?

RQ2 How common are the following ways of organizing work: linear model (waterfall, v-model, etc.), agile, DevOps, BizDev and program organization?

RQ3 Is the realization of benefits perceived to be higher in work organized as project or as CPD?

RQ4 Are there differences in the perceived benefits among those organizing work using linear models, agile, DevOps, BizDev or program organization?

IV. Research Method

We conducted a survey consisting of an online questionnaire to address the research questions. Data was collected during a webinar titled *Digitalization as Continuous Product Development* in June 2021. During this webinar selected IT-professionals presented experiences and reflections on CPD.

Respondents were asked to base all answers on the latest IT system product development they had taken part in (either organized as project or CPD), where the product or part of the product had been taken into use. As discussed in [19], selecting the last project (in this case the last IT product development) "... reduces the risk that the sample of projects is biased towards the most successful or the largest software projects ..." [19]. This is relevant, because we want to compare characteristics of the reported work with perceived success in realizing benefits.

Comparing the sizes of projects and CPD is not straightforward, since CPD does not have a defined end-date, and consequently not a comparable amount of man-hours to compare with projects. Nevertheless, for a description of the size of the work, we asked respondents to provide the number of people involved in the work at most. Then, when including size in the analysis, we used the categories of organization sizes proposed by the European Commission [20].

A. Survey Questions

The survey questions most relevant to answering the research questions are presented in Table I. A complete list of survey questions and responses are available at [21].

TABLE I: SURVEY QUESTIONS

Question	Answer options
SQ1 *How was this work organized?*	Select one: -Project -CPD -Other, explain: [text field]
SQ2 *Approximately how many people was/are actively involved in the work at most? - both from the product owner side and the product developer side*	[text field]
SQ3 *By and large, how well do you consider that you succeeded in realizing benefits as a result of the product?*	Five-point ordinal: (Very Successful 1–5 Very Unsuccessful + 6 Don't know)
SQ4 *Was the product owned by public or private sector?*	Select one: -Public sector -Private Sector
SQ5 *Which ways of working/organizing was used?*	Multiple choice: -Linear model (waterfall, V-model, etc.) -Agile -DevOps -BizDev -Organized as program -Other: [text field]

B. Respondents and Response Rate

A total of 140 people were present at the seminar at the point in time when the survey started. Of these, 131 participated in the survey, but 19 stopped after providing demographic data only. These 19 are not included in the survey results. In total $n_{partial}$=112 (85%) people completed the first three pages of the survey, and $n_{complete}$=94 (72%) finished the entire survey. The size of work reported on are in the following categories: 20% micro (<10 people involved), 55% small (10–49 people involved), 20% medium (50–249 people involved) and 4% large (>250 people involved).

Figure 1 shows an overview of the success in realizing benefits. Compared to what is reported in similar studies (see [19]), it seems that respondents in this survey reported on work that is more successful in realizing benefits.

Among the $n_{partial}$ respondents, 36% represented the product owner side exclusively, 49% represented the product developer side exclusively, while 15% represented both sides. The respondents' average experience with creation of digital solutions ranged from under a year to 50 years, with a mean of 17.6 years and median of 20 years. The number of years

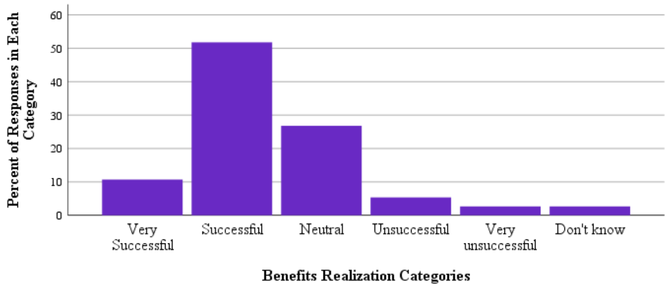


Fig. 1: Degree of Realization of Benefits

of experience as a manager in this field ranged from one to 30 years, with a mean of 10.5 years and median of 9 years. Further, 51.8% of the respondents reported to organize work as projects, while 40.2% reported to organize work as CPD, and 8% reported to organize work as "other" (most of these were combinations of project and CPD). Finally, 68% of the respondents reported on products owned by the public sector, while 32% reported on products owned by the private sector.

C. Analysis

For each of the four research questions, we present descriptive statistics for the corresponding survey questions. For RQ3 and RQ4, where we look at relations between variables, we also present significance values and effect sizes for the comparisons.

When comparing projects and CPD with respect to the realization of benefits (RQ3 and RQ4), we exclude data from work organized as "other" and data where the respondents reported not to know the degree of benefits realization. We use a t-test to calculate significance values and effect sizes [22]. Although the data is not normally distributed, the sample size is large enough for the t-test even with a skewed sample (condition: $n \geq 40$ [22, p. 516], current sample: n_{project}=56, n_{CPD}=45). For effect sizes we use Cohen's d with the following rules of thumb [23]: <0.1 (very small), 0.1 – <0.3 (small), 0.3 – <0.5 (medium), 0.5 – <1.2 (large), 1.2 – <2.0 (very large) and >=2.0 (huge). We use a two-tailed test [22], because we make no assumptions on projects performing better or worse than CPD.

When comparing other ways of organizing work with respect to the realization of benefits, we exclude only the data where respondents reported not to know the degree of benefits realization. Because some of the ways of organizing work have few reported occurrences, we use the Fisher's exact test [24] to calculate significance values.

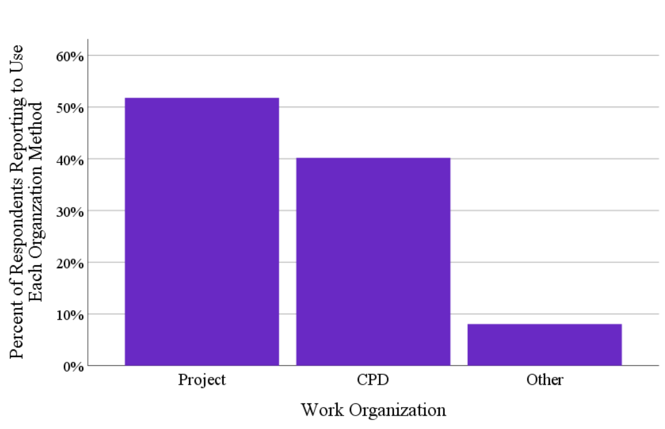
V. Results

A. Organization of work (RQ1 and RQ2)

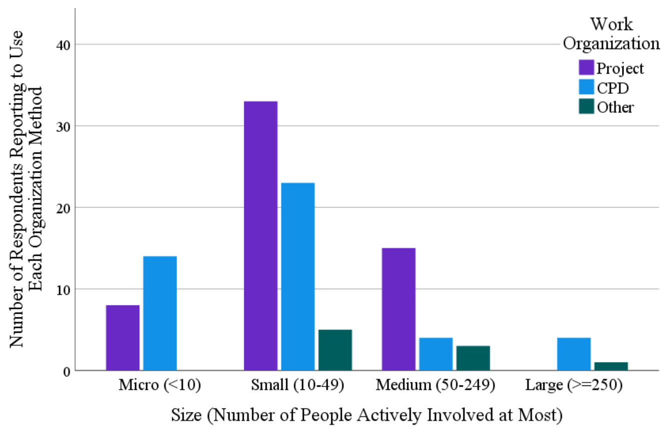
Figure 2 summarizes the results on the different ways of organizing work. Figure 2a shows the data from SQ1, where we see that project organization is more common than CPD. Those selecting the "other"-category, were combining project organization and CPD, or using program organization; often organizing work using CPD-like aspects within a project.

Figure 2b shows work organization (SQ1) differentiated by size (SQ2). We observe that the micro and large endeavors use CPD more often than project organization. When looking at the small- and medium-sized endeavors, project organization is more common than CPD. It is interesting to note that large endeavors observed here are either organized as CPD or using program organization (reported as other, with program organization written in the freetext field).

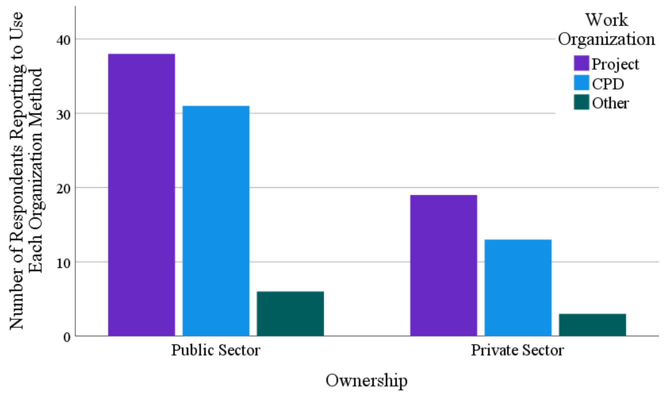
Figure 2c shows differences between endeavors in the public sector versus the private sector (SQ4) in selection of work organization (SQ1). Visual inspection indicates no substantial



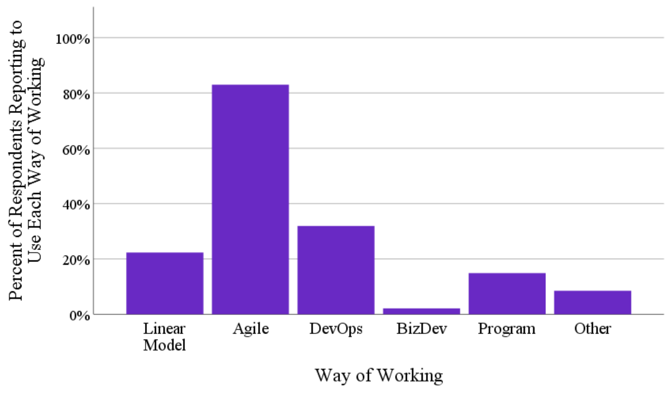
(a) Use of Organization Method (n=112)



(b) Use of Organization Method Grouped by Size (n=110)



(c) Use of Organization Method Grouped by Owner (n=110)



(d) Use of Way of Organizing Work (n=94)

Fig. 2: Organization of work

differences between work organization in efforts owned by the public versus the private sector.

Figure 2d shows which ways of organizing are most commonly applied (SQ5). SQ5 is a multiple choice question, allowing respondents to select several ways of organizing work. Respondents report the use of agile to be very common, with 84% of respondents reporting to use agile in their latest completed endeavor. DevOps is the second most common way of working (32% adoption). Interestingly, linear models are also somewhat frequently used (23% adoption), and 21.5% of those using agile, also use a linear model.

B. Comparing the Use of Practices with Realization of Benefits (RQ3 and RQ4)

Table II shows the realization of benefits (SQ3) when organizing work as projects versus CPD (SQ1), by percentage of responses in each benefits realization category. Our data shows that work organized as CPD is perceived to be significantly (two-tailed t-test p=.020) more successful in realizing benefits than work organized as projects. The effect size of the comparison is Cohen's d=0.475, which is considered a medium effect size. Although this does suggest that CPD outperforms projects, there are other takeaways from Table II worth highlighting:

- 55% of the projects were reported to be successful or very successful (78% for CPD).
- Projects are more distributed on the success-scale than CPD.
- Work organized as CPD was always reported to be neutral or better.

In summary, work organized as CPD was perceived to be more successful than projects, but there are also many successful projects. Among the work reported to use CPD, there were no unsuccessful occurrences.

TABLE II: PERCEIVED REALIZATION OF BENEFITS FOR WORK ORGANIZED AS PROJECT VS CPD (n=101)

Benefits realization	*Project*	*CPD*
Very Successful	∼ 12%	∼ 9%
Successful	∼ 43%	∼ 69%
Neutral	∼ 29%	∼ 22%
Unsuccessful	∼ 11%	0%
Very Unsuccessful	∼ 5%	0%
Two tailed t-test	p=.020	
Effect size	d=0.475	

TABLE III: REALIZATION OF BENEFITS FOR EACH WAY OF ORGANIZING WORK (*n*=93)

	Linear		*Agile*		*DevOps*		*Program*	
Benefits realization	No (n=71)	Yes (n=22)	No (n=14)	Yes (n=79)	No (n=63)	Yes (n=30)	No (n=80)	Yes (n=13)
Very Successful	13%	9%	7%	13%	14%	7%	13%	8%
Successful	58%	50%	57%	56%	51%	67%	55%	62%
Neutral	24%	27%	29%	24%	27%	20%	26%	15%
Unsuccessful	5%	5%	0%	6%	6%	3%	5%	8%
Very Unsuccessful	0%	9%	7%	1%	2%	3%	1%	8%
Fisher's exact test	*p=0.208*		*p=0.594*		*p=0.589*		*p=0.462*	

Table III shows comparisons between the other categories for organizing work (SQ5) with respect to benefits realization (SQ3). Using the Fisher's exact test (bottom row in Table III), we see that our data does not show significant differences in the realization of benefits between those using or not using any of the ways of organizing work. Thus, our data data does not support the current trends in software engineering, where linear models are considered inferior, and agile and DevOps has become very popular.

VI. Discussion

The observation that CPD outperforms projects when it comes to the realization benefits, is relevant for several groups.

First, it is relevant for those making investment decisions. By organizing investment into projects there might be a perceived predictability of time, cost and scope, but this seems to come at he expense of reduced benefits realization.

Second, it is relevant for those working to create software products and realize the benefits of those products. Organizing work in a manner that increases the probability of realizing benefits, helps team members achieving the purpose of their work, which is likely to provide improved job satisfaction.

Third, it is relevant for researchers, because it raises the awareness that several ways of organizing IT development and lifecycle work can lead to success; even though some modes of organizing work are currently in vogue. Researchers should not become evangelists for one approach or the other. Rather, one should conduct research that helps us understand the characteristics of situations where various ways of organizing work – and in particular, project or CPD – is most suitable.

We are puzzled to observe that neither the use, nor non-use, of linear models, agile, DevOps or program organization, had any significant effect on the realization of benefits. Given that further studies with larger power (see next section) corroborate this, one might contrast this to what seems to be the mainstream opinions in the software industry, where linear models are considered bad, and agile is very popular. We speculate that this could be due, either to practitioners making good choices about various ways of organizing work, or, conversely, that practitioners are not successful in using the different ways of organizing work. Both of these situations could explain the lack of differences, and more in-depth studies are called for to unravel the connections between various nuances in ways of organizing work and success in benefits realization. In the mean time one might speculate as follows:

Looking at the results under the assumption that practitioners are making good choices of ways of organizing work, one can speculate that practitioners employ linear models in situations of low uncertainty (where this approach would fit) and agile approaches when there is more uncertainty. This assumption is supported by the observation that the overall degree of realization of benefits reported here, is higher than in similar studies [19] (see Section IV-B and Figure 1).

Alternatively, looking at the results from the view that practitioners are not able to utilize the different ways of organizing work, could help explain why using agile, DevOps and program organization does not seem to lead to higher realization of benefits. This view finds support in challenges reported when introducing agile, DevOps and program organization. Introducing new ways of working can be a challenging task [25], including resistance to change [26] and pressure to use traditional approaches [27]. For DevOps, there is lacking consensus on the best way of organizing work to ensure collaboration between development and operations [28].

VII. Limitations

A. Statistical Conclusion Validity

The low number of respondents (112) in this survey, gives low statistical power with a low probability of observing significant results that are actually present in the population. Replicating studies using a larger sample may find effects that were not uncovered in our data.

It is possible that a webinar on CPD attracts people who are already using CPD. If this is the case, the percentage of respondents using CPD would be higher here than in the population of software product development initiatives. However, the resulting near equal group sizes for projects versus CPD was beneficial for the purpose of answering our research questions via the present survey.

B. External Validity

Based on demographic data in this study, one can generalize the results to populations with similar characteristics. For our sample, this can be problematic for three reasons:

1) We have limited demographic information of the work reported on in this study. Limited information was collected due to time limitations duration the webinar.
2) It is possible that practitioners attending a seminar on CPD think differently about organization of software work than others. As such, it is likely that the number of respondents reporting to organize work using CPD is

higher in our sample than in the population of software development endeavors in the industry.

3) A webinar and survey conducted in Norwegian, limits the effects of cultural difference, and difference in respondents background. It is possible that respondents with a different background would yield different results.

C. Construct Validity

A recent study of the use of linear models and agile methodology [29] found that work organization only accounts for 40% of observed activities in organizations, while the remaining 60% are a result of method incompleteness, peoples skills and habits, organizational noise and similar factors. Thus, when practitioners report to use a linear model, agile, DevOps, BizDev or program organization, it is not clear exactly to what extent, or how, these ways of organizing are implemented.

We observed that the number of respondents reporting to use BizDev was very small. Our impression after talking with practitioners, is that many of the practices in BizDev are commonly used. If this is the case, it seems BizDev is not known to practitioners, resulting in a low count in the survey.

VIII. Conclusion and Further Research

Based on the above results and discussion, we conclude that CPD is a very viable alternative to organizing software investments as projects, especially in situations where failure to realize benefits must be avoided. Also, we believe that more research is needed to understand in what situations practitioners would benefit from selecting eithe project or CPD organization, or a combinaton of both.

We did not find any evidence for or against the use of linear models, agile, DevOps, BizDev or program organization. Still, we believe that all of these has previously been used to realize the benefits of software investments successfully. We suggest that more research is needed to understand better these ways of organizing work, especially focusing on how the different ways of organizing work perform in different situations. This we hope will provide practitioners with actionable guidance on the selection of ways of organizing work.

Acknowledgments

The authors are grateful to the survey respondents for their time and experience.

References

[1] R. Berntsson-Svensson and A. Aurum, “Successful software project and products: An empirical investigation,” in *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pp. 144–153, 2006.

[2] S. Goldfinch, “Pessimism, computer failure, and information systems development in the public sector,” *Public Administration Review*, vol. 67, no. 4, pp. 917–929, 2007.

[3] A. Bharadwaj, M. Keil, and M. Mähring, “Effects of information technology failures on the market value of firms,” *The Journal of Strategic Information Systems*, vol. 18, no. 2, pp. 66–79, 2009.

[4] K. Conboy, “Project failure en masse: a study of loose budgetary control in isd projects,” *European Journal of Information Systems*, vol. 19, no. 3, pp. 273–287, 2010.

[5] D. Baccarini, “The logical framework method for defining project success,” *Project management journal*, vol. 30, no. 4, pp. 25–32, 1999.

[6] E. Bouwers, J. Visser, and A. Van Deursen, “Getting what you measure,” *Communications of the ACM*, vol. 55, no. 7, pp. 54–59, 2012.

[7] S. S. Tanilkan and J. E. Hannay, “Benefit considerations in project decisions,” in *International Conference on Product-Focused Software Process Improvement*, pp. 217–234, Springer, 2022.

[8] R. Kneuper, “Sixty years of software development life cycle models,” *IEEE Annals of the History of Computing*, vol. 39, no. 3, pp. 41–54, 2017.

[9] T. E. Bell and T. A. Thayer, “Software requirements: Are they really a problem?,” in *Proceedings of the 2nd International Conference on Software Engineering*, pp. 61–68, 1976.

[10] T. Thesing, C. Feldmann, and M. Burchardt, “Agile versus waterfall project management: Decision model for selecting the appropriate approach to a project,” *Procedia Computer Science*, vol. 181, pp. 746–756, 2021.

[11] W. Royce, “Managing the development of large software systems: concepts and techniques,” in *Proceedings of the 9th International Conference on Software Engineering*, pp. 328–338, 1987.

[12] C. Larman and V. Basili, “Iterative and incremental developments. a brief history,” *Computer*, vol. 36, no. 6, pp. 47–56, 2003.

[13] J. Bosch, *Continuous software engineering*, ch. Continuous Software Engineering: An Introduction, pp. 3–13. Springer International Publishing, 01 2014.

[14] L. Williams and A. Cockburn, “Agile software development: it’s about feedback and change,” *Computer*, vol. 36, no. 6, pp. 39–43, 2003.

[15] M. Fowler *et al.*, “The agile manifesto,” *Software development*, vol. 9, no. 8, pp. 28–35, 2001.

[16] K. K. Holgeid and M. Jørgensen, “Benefits management and agile practices in software projects: how perceived benefits are impacted,” *IEEE 22nd Conference on Business Informatics (CBI)*, vol. 2, pp. 48–56, 2020.

[17] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “Devops,” *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[18] B. Fitzgerald and K.-J. Stol, “Continuous software engineering: A roadmap and agenda,” *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.

[19] M. Jørgensen, “A survey on the characteristics of projects with success in delivering client benefits,” *Information and Software Technology*, vol. 78, pp. 83–94, 2016.

[20] European Commission and Directorate-General for Internal Market, Industry, Entrepreneurship and SMEs, *User guide to the SME definition*. Publications Office, 2020.

[21] S. S. Tanilkan, “Organization as project vs continuous product development.” https://tinyurl.com/pvscpd. Accessed: 2023-04-24.

[22] D. S. Moore and G. P. McCabe, *Introduction to the Practice of Statistics*. W. H. Freeman and Company, 2001.

[23] S. S. Sawilowsky, “New effect size rules of thumb,” *Journal of Modern Applied Statistical Methods*, vol. 8, no. 2, pp. 596–599, 2009.

[24] S. Weerahandi, *Exact statistical methods for data analysis*. Springer Science & Business Media, 2003.

[25] Z. Shehu and A. Akintoye, “Major challenges to the successful implementation and practice of programme management in the construction environment: A critical analysis,” *International Journal of Project Management*, vol. 28, no. 1, pp. 26–39, 2010.

[26] T. J. Gandomani, H. Zulzalil, A. A. Ghani, A. B. M. Sultan, and K. Y. Sharif, “How human aspects impress agile software development transition and adoption,” *International Journal of Software Engineering and its Applications*, vol. 8, no. 1, pp. 129–148, 2014.

[27] C. de O. Melo *et al.*, “The evolution of agile software development in brazil,” *Journal of the Brazilian Computer Society*, vol. 19, no. 4, pp. 523–552, 2013.

[28] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, “A survey of devops concepts and challenges,” *ACM Comput. Surv.*, vol. 52, pp. 1–35, Nov. 2019.

[29] B. V. Thummadi and K. Lyytinen, “How much method-in-use matters? a case study of agile and waterfall software projects and their design routine variation,” *Journal of the Association for Information Systems*, vol. 21, no. 4, pp. 863–900, 2020.

A Model Library Tool for Holistic Embedded Software Design

Sven Jacobitz, Xiaobo Liu-Henke
Ostfalia University of Applied Sciences
Department of Mechanical Engineering, Institute for Mechatronics
Salzdahlumer Str. 46/48, 38302 Wolfenbüttel, Germany
Email: {sve.jacobitz; x.liu-henke}@ostfalia.de

***Abstract*—The ever-increasing complexity and connectivity of mechatronic systems makes using a structured, systematic methodology essential for embedded software design. The model-based Rapid Control Prototyping is a widely used model-based development process for this. Essential is a seamless support by a Computer Aided Engineering platform. However, such platforms are very cost-intensive, which is why the seamless low-cost platform LoRra was developed by the authors. A key element of this platform is the Model Library tool, which provides consistent access and traceable change management to all data (especially functional and plant models as well as resulting artefacts) used throughout the holistic process of software development. Version and configuration management also increase the reusability of resulting artefacts. This paper presents the new ideas, used to design the LoRra model library for the low-cost function development of mechatronic systems. The holistic coverage of the entire development process, from modelling to real-time realization, is the feature, that distinguishes the LoRra model library from existing tools.**

***Index Terms*—Rapid Control Prototyping (RCP), low-lost development platform, model-based design, model library.**

I. Introduction

Mechatronic systems continue to increase in complexity and functionality. This trend is a major challenge for Small and Medium-sized Enterprises (SMEs). To remain competitive, they need to integrate ever more intelligent hardware and software into their products. This is not only due to the number of functions in a system, but also to the ever-increasing degree of connectivity between complex software components that strongly interact with each other [1]. The structured and systematic development of such software intensive systems is essential to meet ever shorter development times and higher quality requirements [2]. In this context, model-based Rapid Control Prototyping (RCP) is a widely used methodology for embedded software design. The seamless support by a Computer Aided Engineering (CAE) development platform is essential for RCP in order to achieve a high degree of automation. Established seamless CAE tool chains are very cost-intensive, which is a major barrier to the adoption of the RCP process, especially for SMEs [3]. Therefore, as part of the EU-funded research project *Low-Cost Rapid Control Prototyping System with Open-Source-Platform for Functional Development of Embedded Mechatronic Systems (LoCoRCP)*, the authors developed a seamless low-cost development platform named LoRra [4].

This paper presents the conception, design and exemplary realization of the CAE-based LoRra model library. This library enables access to a consistent data base throughout the entire development process, as well as traceable change management - especially for functional or plant models and resulting artefacts. The rest is structured as follows: Section III summarizes the RCP development methodology and introduces the LoRra platform. The state of the art is outlined in Section II. In Section IV, the concept and the basic solution approach is presented based on a requirement analysis. The design is detailed in Section V. Section VI is a description of the implementation. Finally, Section VII summarizes the results and gives an outlook on future work.

II. State of the art

Developing in distributed teams and the associated central data management has been an important topic in classical software development for a long time. This is especially, due to the high diversity, flexibility and short development times of software [5]. Model-based software development poses new challenges for methods and tools. To ensure the reusability of models and the associated software functions newly, systematic, integrated data management approaches with the associated sub-processes, such as version and configuration management must be used. This is important because, in comparison to classical software development, the resulting artefacts no longer result from manual textual changes, but are derived from models [6].

First regulations for this came up in the early 1960s at NASA [7]. According to Sax et al. [8], inconsistencies during function development are increasingly caused by the high number of variants, which can be avoided by using an appropriate configuration management. In the context of RCP, a CAE-based model library that manages all relevant artefacts (result of a subprocess such as models, program source code or documentation) is recommended for this purpose [9].

Version and configuration management is widely used in classical software development. An overview is provided by [10]. A primitive but widely used method is manual versioning. A backup is generated by manual copying and renaming. In comparison, the backup copies are created automatically when version control tools are used. There are many different kinds of software available, such as the Concurrent Version System (CVS), Subversion (SVN) or Mercurial.

A frequently used open source tool for versioning is GIT [11]. However, the focus of this tool is on change-based management of text files [12]. An application of this approach to data formats common in model-based design is not very practicable [13]. Therefore, extensive adaptations are required for use in a model library.

In order to identify the type and scope of the necessary adaptations, systematic investigations were carried out by Niedzwiedz and Frei [14], for example. Here, a model is constructed in a standardized way from metadata, interface information and parameters. Based on such a standardized structure, version and configuration management can be performed even for complex, integrated models. An example of such a systematic structural description approach is the System Entity Structure (SES) [15].

In summary, there is currently no applicable solution for central model management within the framework of a low-cost RCP platform. Available approaches are either designed for textual changes, but do not offer the structures necessary for model-based software design or do not support essential processes, such as version and configuration management. Approaches that exist for UML-based models, for instance (c.f. [16]), often only support the models themselves, but not the resulting artefacts such as the source code. Also, the holistic coverage of the entire development process, from modelling to real-time realization, is not supported by other tools, yet.

III. Development methodology and platform

Due to the high system complexity of modern interconnected mechatronic systems, the structured, model-based, verification-oriented RCP process is used for software development and validation. This consists of the process steps modelling, analysis / synthesis, automated generation of source code, automated implementation on real-time hardware and online experimentation. The whole methodology is supported by Model-in-the-Loop (MiL), Software-in-the-Loop (SiL) and Hardware-in-the-Loop (HiL) simulations [17].

The presented methodology is characterized by a high degree of consistency and automation, from modelling and model-based software design to automatic code generation and real-time realization (cf. Fig. 1 on the left). It is accompanied by a seamless, fully automated CAE platform. The modular, cost-effective development platform LoRra is such a CAE platform. Fig. 1 illustrates the RCP development process, as well as the seamless support by means of LoRra [4]. Of particular relevance here is a central model library that makes a consistent, traceable development status available in all process steps.

The domain-independent model library serves as a central data base from the modelling process up to the realization. By means of version and configuration management, model variants can be designed, managed and integrated to higher level models. The open source CAE tool Scilab / Xcos (cf. [18]) is used for model analysis and synthesis of the

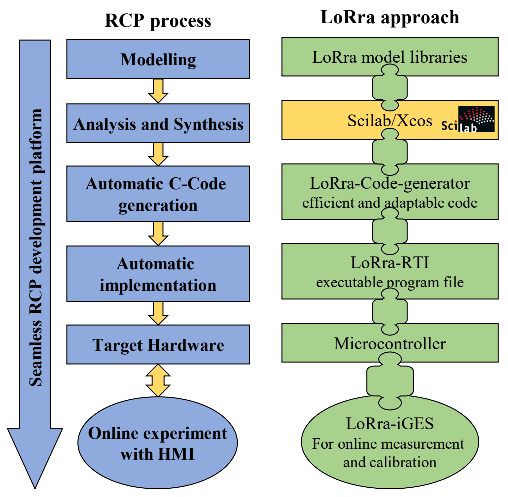


Fig. 1. RCP development process with seamless support by the LoRra platform [4].

software. It offers a wide range of functionality comparable to those of the commercially frequently used Matlab / Simulink. The resulting function model can be integrated directly into the model library. Thanks to the open interfaces of the LoRra API, existing programmes and interface drivers can also be integrated with little effort. MiL simulations can be used to optimize and test the developed functions at an early stage of development.

The LoRra code generator automatically generates efficient, modular C source code from the functional model by means of model-to-text transformation. Open functional descriptions of basic elements of the model, so-called basic blocks, make the LoRra code generator flexibly extendable. The generated source code can be re-integrated into the Xcos model without manual work, e.g., for optimization and testing by means of SiL simulations.

The signals to the plant models or, depending on the development focus, to other software components are replaced by interface blocks of the LoRra Real-Time Interface (RTI) when the development reaches a sufficient functional status. This enables the use of real-time hardware interfaces without manual programming. In combination with hardware-specific RTI basic software, which includes a real-time operating system and standardised interface drivers, automated implementation on the real-time hardware by the RTI is possible. Low-cost microcontrollers, e.g., of the STM32H7 series, are used as real-time hardware. By means of HiL simulations, the developed function can thus also be optimized and tested under real-time conditions. The integrated Ggraphics-supported Experimentation Software (iGES) is available as a Human-Machine Interface (HMI). It can be used to intuitively perform and monitor online experiments, as well as to record measurement data by means of real-time data acquisition.

IV. Concept of the model library

In this section, the conception of the LoRra model library is presented. First, some approaches for the graphical user interface development are introduced. Then, the requirements are outlined and the initial stage for a solution is derived.

A. Human Machine Interface development approaches

Nowadays, standardised architecture styles are used for structured software design [19]. These serve in particular to increase reusability, to structure the design and to create a uniform vocabulary. More than 25% of the existing styles are used for the design of HMI [20]. In the context of this work, the Model-View-Controller (MVC) principle is particularly relevant.

The architectural style MVC, according to [21], is illustrated in Fig. 2. Here, the view (also called visualization or presentation), the controller and the data model are realized separately with defined interfaces. The controller component reacts to user inputs in the Graphical User Interface (GUI) and changes the model if necessary. Furthermore, the model can also be changed by other software components. It notifies the controller of the changes made. The controller updates the presentation. Due to the low component coupling, this principle is particularly suitable for HMI that are used on different target platforms [22]. For example, the operating system-dependent graphical presentation can be completely decoupled from the controller and the model.

B. Model library requirements

As outlined in Section III, the model library is a central tool for data management in all process steps of software development. In order to fully support the model-based version and configuration management approach, the model library must meet the following overarching requirements:

1) Coherent versioning of all contained data and support for version management processes (e.g., review, approval).
2) Supporting the data structures required for Configuration Management throughout the process steps, as well as the Configuration Management processes (e.g. review, approval) to ensure a consistent data state at all times and across all RCP steps.
3) Hierarchical structuring of the models in configurable categories and hierarchy levels.
4) Search function to quickly find specific models, even though a large amount of data are contained.
5) Support for distributed teams working from a common model base.
6) Presentation of all relevant model information in one overview.

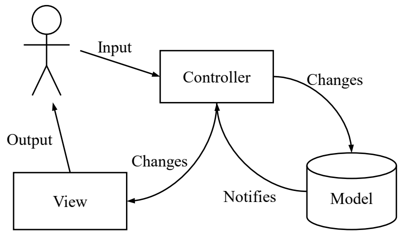


Fig. 2. Architecture style Model-View-Controller according to [21].

C. Basic idea of the concept

In order to be fully compliant with the requirements of the model library, the first step is a closer look at the structure of a model. In doing this, generic and aggregated models need to be distinguished.

A generic model is the smallest self-contained model unit at the lowest hierarchical level. It is not subdivided into further hierarchically ordered sub-models. An example of a generic model is the electrical part of a DC motor, which can be described by (1) (cf. [23]). The structured assembly of a generic model is illustrated by Fig. 3. It consists of four components:

- **Metadata** describes the higher-level characteristics (e.g., name, author, general description) of the model.
- **Interface information:** Data structure, units and other relevant information of the inputs- and outputs of the model. Using the example of (1), the terminal voltage u in V and the angular speed ω in rad/s as input or the motor current i in A as output.
- **Parameter:** Information and values about the parameters of the model. Using (1) as an example, the resistance R in Ω, the inductance L in H and the machine constant c in Vs.
- **Artefacts** of the model such as the (Xcos-)model file, the generated C-code or the model documentation.

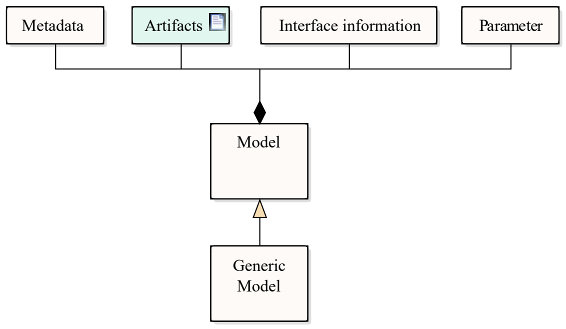


Fig. 3. Structured assembly of a generic model.

$$u = Ri + L\frac{\mathrm{d}i}{\mathrm{d}t} - c\omega \tag{1}$$

An aggregated model is composed of further sub-models and thus represents higher hierarchy levels. Aggregated models are mapped as so-called configurations in the model library. A configuration is created by integrating defined version levels of the part models. Fig. 4 illustrates the principle assembly of a configuration.

The models are arranged hierarchically in a tree structure. The tree contains folders (grouping hierarchy elements) and model elements (both generic and aggregated models). User rights and individual processes can be assigned to both groupings and individual elements.

To enable model access for several users, the principle of a central data repository is applied. Fig. 5 illustrates the concept. All model data are stored in the central repository, which acts

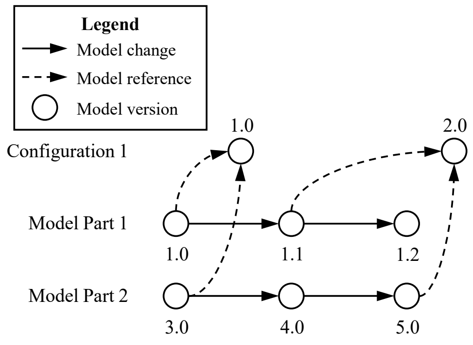


Fig. 4. Principle assembly of a configuration.

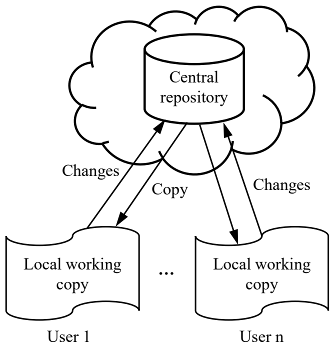


Fig. 5. Concept of the central data repository.

as a database. Model artefacts are accessed via local working copies. Changes are transmitted to the central repository and merged to the original data. Users can then pull the changed data to their local working copy.

V. Model library design

The concept from Section IV will now be fleshed out and transferred into a concreted design. For this purpose, the data structures and interfaces, as well as the data management are designed.

A. Data structures

There are a number of data structures and interfaces that are necessary for the model library. The core element is a hierarchical model tree, which also serves as the data basis for the MVC of the GUI. In the following, the data structure and interfaces of the model tree are designed as an example.

The set-up of the data structure is object-oriented. For each element of the tree, the abstract class *AModelElement* represents the basic structure. It contains central data such as title, path in the tree or parent element. The classes *HierarchyElement* and *ModelElement* are derived from it. *HierarchyElement* contains a list with subordinate elements. *ModelElement* summarizes interface information, parameters and memory information of the model among other data. Fig. 6 illustrates the relationship as a UML class structure.

The model tree requires interfaces for various operations, which are served by the controller classes. For example, adding or moving child elements of the class *HierarchyElement* is

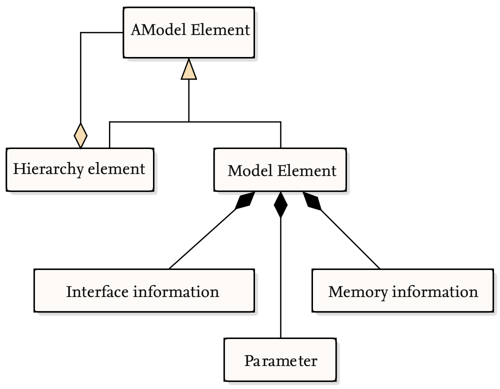


Fig. 6. Class structure of the hierarchical model tree.

provided here. Model elements require more extensive interfaces, for example to change the metadata or to generate new versions. Each modification must be captured and documented by the version management.

B. Data Management

Data management is the key function of the model library. On the one hand, version and configuration management are very important. On the other hand, the storage of local working copies must also be coordinated. For example, only models that have been selected by users should be completely downloaded as working copies. For other models, storing the metadata is enough.

To ensure versioning and thus consistent reuse of software in the form of configurations, the model library must support an appropriate version management process. This mainly concerns releasing new versions. If changes are made to a model or other artefacts, a new model version may only be used following specific release processes. For the LoRra model library, this means that versions proposed by users are not released to the public until they have been approved by the groups of people required according to the configured process.

For version numbering, the concept of semantic versioning is applied. Different versions of a models and artefacts are identified by version numbers of the form x.y. Where x is the major version and y is the minor version. If no compatibility-relevant adjustments were made to the model during a change (e.g., bug fixes or pure visual changes - behaviour and interfaces remain the same), only the minor version is incremented. If adjustments have been made that affect the compatibility of the model with other models (e.g., changes to the interfaces, extension of the functionality), the major version is incremented and the minor version is set to 0.

The composition of a configuration is done by linking submodels. For this purpose, the corresponding version numbers of the models are referenced, and the interfaces are linked. Fig. 4 illustrates the principle.

VI. Realization

The model library is implemented in Java as Eclipse Rich Client Platform (cf. [24]). A basic set of functions is implemented in an object-oriented way. The Eclipse framework already offers many mechanisms necessary for realization, such as the *Standard Widget Toolkit* or event-based, minimal-coupling communication between different graphical elements. In addition, numerous extensions with open interfaces can be used.

Versioning is done using the existing open source tool GIT (cf. [11]), which also connects to the central storage infrastructure. There is a separate GIT repository for each model. Proven mechanisms for versioning are already available here. By using structured, text-based model descriptions, the limitations mentioned in Section II can be avoided. The GIT branching enables variant management in addition to the version and configuration management functions described above. Initially, user authentication is implemented for the Atlassian service Bitbucket. Later extension is possible.

The structured model description is in JSON format (cf. [25]). Listing 1 contains an exemplary stored model tree. Hierarchy elements are identified by the fields *title* (display title of the element), *relPath* (relative file path to the parent hierarchy element) and *children*. Model elements contain the fields *relPath*, *metaFileName* and *repoUrl* (URL to the online GIT repository). All relevant metadata is stored in the file specified in *metaFileName*.

The integration of sub-models into a configuration is XML-based. This is done in the form of an SES. Thus, the structure of a new configuration can first be created at an abstract level. A concrete configuration is then generated by pruning and referencing the sub-models and specifying the version and variant. This approach with flexible, standardised data structures and interfaces allows the model library to be applied in various simulation environments. It can therefore be used as part of the LoRra platform, as well as in the context of other development platforms or further model editors.

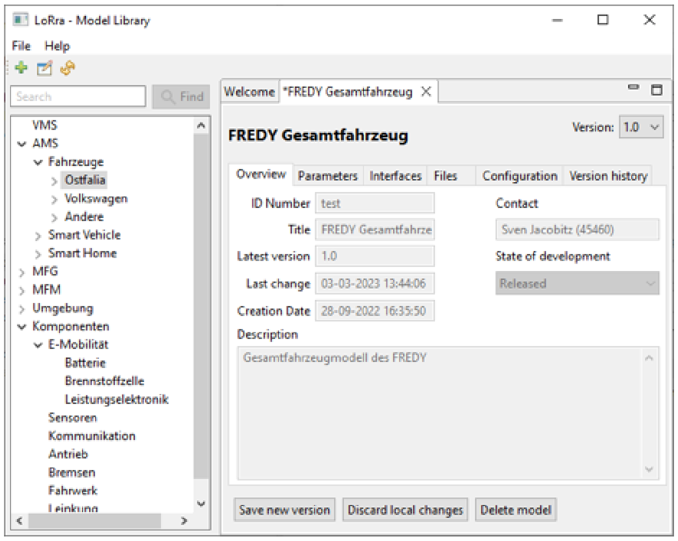
Finally, the GUI is built according to the MVC principle introduced in Section IV-A. The data basis for this (model) is the model tree designed in Section V-A. Fig. 7 illustrates the overall design (presentation) of the LoRra model library. The main window is divided into three areas. The navigation area (on the left) contains the hierarchical model tree of the library. Here, users can perform actions on individual models (e.g., open or edit) and get an initial overview of the current model status. In addition, the model tree can be searched. The display area (on the right) contains various views for displaying and editing information. Here, for example, the metadata and model artefacts can be displayed or the version history can be viewed. In addition, a context-dependent toolbar and the menu structure for operating the library are arranged in the tool area.

VII. Summary and Future Work

This paper presents the design of a model library for low-cost software development of mechatronic systems using model-based design approaches. As part of the seamless RCP development platform LoRra, which is based on open source software, the model library provides a consistent and traceable model basis for each development step. In this way, the model

Listing 1. Exemplary model tree in JSON format.

```
{
  "title" : "root",
  "relPath" : "",
  "children" : [ {
    "title" : "Vehicle models",
    "relPath" : "vehicles/",
    "children" : [ ... ]
  }, {
    "title" : "Functional models",
    "relPath" : "functions/",
    "children" : [ {
      "title" : "VMS",
      "relPath" : "VMS/",
      "children" : [ ... ]
    },
    {
      "title" : "AMS",
      "relPath" : "AMS/",
      "children" : [ {
        "relPath" : "efm/",
        "metaFileName" : "efm.json",
        "repoUrl" : "https://tinyurl.com/
            repo_efm/"
      }, ... ]
    } ]
  }, ... ]
}
```



FREDY Gesamtfahrzeug

Version: 1.0

ID Number: test
Title: FREDY Gesamtfahrze
Latest version: 1.0
Last change: 03-03-2023 13:44:06
Creation Date: 28-09-2022 16:35:50
Contact: Sven Jacobitz (45460)
State of development: Released
Description: Gesamtfahrzeugmodell des FREDY

Fig. 7. Layout of the graphical user interface of the model library.

library covers the entire development process holistically. Based on the basic requirements, an approach for version and configuration management of hierarchical models, as well as for central model storage was developed. This was followed by the exemplary design of a model tree data structure for hierarchical model configurations and the graphical user interface. Finally, the implementation of a basic functionality is summarized.

Based on existing technologies, such as version management with GIT, a new tool for the continuous development of software for mechatronic systems has been created. An easy-to-use graphical interface facilitates version and configuration management of project artefacts throughout the entire development process from MiL, SiL and HiL to prototype.

Future work will focus on further optimizing the user experience. To this end, the integration of graphical editors to simplify the generation and management of configurations is also possible. An extension of the GIT tool *diff*, which visualizes model changes, is planned for an optimized overview of the version history. Finally, a generalization of user authentication is possible, so that any kind of central storage system can be used. For further testing and optimization, the model library will be integrated into the virtual embedded software test bench of the authors.

Acknowledgment

Funded by the Lower Saxony Ministry of Science and Culture under grant number ZN3495 within the Lower Saxony "Vorab" of the Volkswagen Foundation and supported by the Center for Digital Innovations (ZDIN).

ZDIN
Zentrum für digitale Innovationen Niedersachsen

VolkswagenStiftung

Niedersächsisches Ministerium für Wissenschaft und Kultur

References

[1] X. Liu-Henke, S. Scherler, M. Fritsch, and F. Quantmeyer, "Holistic development of a full-active electric vehicle by means of a model-based systems engineering," in *Proceedings of 2016 IEEE International Symposium on Systems Engineering (ISSE)*, B. Rassa and P. Carbone, Eds., 2016, pp. 1–7.

[2] S. Jacobitz, M. Gollner, J. Zhang, O. A. Yarom, and X. Liu-Henke, "Seamless validation of cyber-physical systems under real-time conditions by using a cyber-physical laboratory test field," in *2021 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*. IEEE, 2021, pp. 1–8.

[3] X. Liu-Henke, R. Feind, M. Roch, and F. Quantmeyer, "Investigation of low-cost open-source platforms for developing of mechatronic functions with rapid control prototyping," in *Proceedings of the 2014 International Conference Mechatronic Systems and Materials (MSM)*, 2014, pp. 1–9.

[4] S. Jacobitz and X. Liu-Henke, "The Seamless Low-cost Development Platform LoRra for Model based Systems Engineering," in *Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS - Science and Technology Publications, 2020, pp. 57–64.

[5] H.-B. Kittlaus, *Software Product Management*. Berlin, Heidelberg, Germany: Springer, 2022.

[6] K. Henderson and A. Salado, "Value and benefits of model–based systems engineering (MBSE): Evidence from the literature," *Systems Engineering*, vol. 24, no. 1, pp. 51–66, 2021.

[7] B. L. Summers, "Software Configuration Management," in *Effective Methods for Software Engineering*, B. L. Summers, Ed. New York, USA: Auerbach Publications, 2020, pp. 57–65.

[8] H. Guissouma, H. Klare, E. Sax, and E. Burger, "An Empirical Study on the Current and Future Challenges of Automotive Software Release and Configuration Management," in *44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 298–305.

[9] B. Kruse and K. Shea, "Design Library Solution Patterns in SysML for Concept Design and Simulation," *Procedia CIRP*, vol. 50, pp. 695–700, 2016.

[10] N. Ratti and P. Kaur, "Case Study: Version Control in Component-Based Systems," in *Designing, Engineering, and Analyzing Reliable and Efficient Software*, H. Singh and K. Kaur, Eds. Hershey, USA: IGI Global, 2013, pp. 283–297.

[11] H. Eriksson, J. Sun, V. Tarandi, and L. Harrie, "Comparison of versioning methods to improve the information flow in the planning and building processes," *Transactions in GIS*, vol. 25, no. 1, pp. 134–163, 2021.

[12] Y. S. Nugroho, H. Hata, and K. Matsumoto, "How different are different diff algorithms in Git?" *Empirical Software Engineering*, vol. 25, no. 1, pp. 790–823, 2020.

[13] D. Schmitz, W. Deng, T. Rose, M. Jarke, H. Nonn, and K. Sanguanpiyapan, "Configuration Management for Realtime Simulation Software," in *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2009, pp. 229–236.

[14] S. Niedzwiedz and S. Frei, "A structured model library for the analysis of electric-vehicle drivetrains," in *AmE 2012 - automotive meets electronics*, ser. GMM technical report. VDE-Verlag, 2012, pp. 21–26.

[15] U. Durak, T. Pawletta, H. Oguztuzun, and B. P. Zeigler, "System entity structure and model base framework in model based engineering of simulations for technical systems," in *Proceedings of the Symposium on Model-driven Approaches for Simulation Engineering*, A. D'Ambrogio, Ed. ACM Society for Computer Simulation International, 2017, pp. 1–10.

[16] R. S. Bashir, S. P. Lee, S. U. R. Khan, V. Chang, and S. Farid, "Uml models consistency management: Guidelines for software quality manager," *International Journal of Information Management*, vol. 36, no. 6, 2016.

[17] X. Liu-Henke, S. Jacobitz, S. Scherler, M. Göllner, O. Yarom, and J. Zhang, "A Holistic Methodology for Model-based Design of Mechatronic Systems in Digitized and Connected System Environments," in *Proceedings of the 16th International Conference on Software Technologies*, H.-G. Fill, M. van Sindern, and L. Maciaszek, Eds. SCITEPRESS - Science and Technology Publications, 2021, pp. 215–223.

[18] A. K. Verma and R. Verma, *Introduction to Xcos - A Scilab Tool for Modeling Dynamical Systems*, 1st ed. Jodhpur, India: MBM Engineering College, JNV University, 2020.

[19] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 3rd ed., ser. SEI series in software engineering. Upper Saddle River, USA: Addison-Wesley, 2013.

[20] S. Henninger and V. Corrêa, "Software pattern communities: current practices and challenges," in *Proceedings of the 14th Conference on Pattern Languages of Programs - PLOP '07*, A. Aguiar and J. Yoder, Eds. ACM Press, 2007, pp. 1–19.

[21] S. Adams, "MetaMethods: The MVC paradigm," *HOOPLA!*, vol. 1, no. 4, 1988.

[22] Z. Liu, F. Li, H. Liu, C. Wu, and J. Zhang, "A Study of Cockpit HMI Simulation Design Based on the Concept of MVC Design Pattern," in *Proceedings of the 2018 3rd International Conference on Modelling, Simulation and Applied Mathematics (MSAM 2018)*. Atlantis Press, 2018, pp. 82–84.

[23] X. Liu-Henke, M. Gollner, M. Fritsch, R. Feind, and R. Buchta, "FreDy - An electric vehicle with intelligent chassis-control systems," in *2015 Tenth International Conference on Ecological Vehicles and Renewable Energies (EVER)*. IEEE, 2015, pp. 1–8.

[24] L. Vogel and M. Milinkovich, *Eclipse Rich Client Platform: The complete guide to Eclipse application development*, 3rd ed., ser. Vogella series. Hamburg, Germany: Vogella, 2015.

[25] "ISO/IEC 21778:2017: Information technology — The JSON data interchange syntax," International Organization for Standardization.

A Lightweight Method to Define Solver-Agnostic Semantics of Domain Specific Languages for Software Product Line Variability Models

Camilo Correa Restrepo
Centre de Recherche en Informatique (CRI)
University of Paris 1 Panthéon-Sorbonne
Paris, France
email: camilo.correa-restrepo@univ-paris1.fr

Raul Mazo
Lab STICC
ENSTA Bretagne
Brest, France
email: raul.mazo@ensta-bretagne.fr

Andres López
Investigación y desarrollo
SoftControlWeb
Medellin, Colombia
email: andresorlandolopez@gmail.com

Jacques Robin
Learning, Data and Robotics Laboratory, ESIEA, Paris, France
Center for Research in Informatics (CRI), University of Paris 1 Panthéon-Sorbonne, Paris, France
email: jacques.robin@esiea.fr

***Abstract*—We propose a method to address the current lack of standards for both software product line variability modeling languages and their formal semantics. It allows specifying, in an agile, declarative, and solver-agnostic fashion the formal semantics of a domain-specific variability modeling language through a simple JSON based specification format. Our approach leverages the Common Logic Interchange Format (CLIF) standard for interoperability among logical inference engines. We demonstrate our approach with two concrete examples of Variability Models, and present the tooling and architecture that makes this possible.**

Keywords*—*Variability Modeling; Formal Semantics; Modeling Language Specification; Common Logic.

I. Introduction

Software Product Lines Engineering (SPLE) [1] is a method to systematically coordinate and automate the engineering and evolution of a large set of related software products with overlapping functionalities and reusable software assets over long life-cycles, whose products, put together, form a ***Software Product Line (SPL)***. In model-driven SPLE, these assets are both models and code files while, in code-driven SPLE, these assets are mostly code files implementing services, components, classes, decorators, aspects and functions. The key artifact that distinguishes an SPL from a single software product is its ***Variability Model (VM)***. It explicitly identifies sets of requirements, generally called *features*, that are cohesive from a business or technical perspective and determines how they partially overlap across the different products of the line. This VM generally organizes those features into an abstraction and composition hierarchy and associates the lowest level ones with reusable and composable concrete software assets implementing them. Developing an SPL VM and such composable assets requires a large upfront investment. However, once done, it enables the automated generation of a very large number of product variants, which, in turn, supports the simultaneously low-cost and rapid delivery of very many customized software products, all maintained and evolved in coordination. It has provided great returns on investment mostly within industries such as transportation, healthcare and energy [2] where systems have a long lifecycle and have a critical nature.

Initially, an SPL VM was a purely design-time artifact used to interactively choose a valid set of features and attribute value choices to then generate the source code of a product variant (*a.k.a.* an SPL configuration) resulting from these choices by composing and/or transforming the associated SPL's reusable assets. More recently, they have started to be used as ***Models-at-Run-Time (M@RT)*** [3] artifacts for context-aware self-adaptive systems that continuously monitor their execution context for changes that might require a run-time reconfiguration. In this approach, called *dynamic* SPLE [4], an additional context model needs to be included into the SPL VM and the whole SPL and its configuration tool are embedded into each deployed product they generate. When monitoring systems detect that in a new context, the current configuration no longer satisfies some system requirements, it triggers the SPL configuration tool to search the SPL VM for alternative configurations better adapted to this new context and then update the implementation with it. To contrast them from dynamic SPLs, the original, purely design-time SPLs are called *static* SPLs.

In the current state of the art, there is no accepted standard for SPL VMs, so every SPLE tool uses its own ***Domain Specific Language (DSL)*** to model the VM. Nonetheless, almost all of these languages used for VMs share four key expressive capabilities. The first is to distinguish between mandatory and optional elements. The second is to specify ranges of alternative possible values for a given element parameter. The third is to specify ranges of alternative possibilities for the refinement of a higher-level elements into a set of lower-level elements. The fourth is to specify complex business and regulatory constraints concerning the co-occurrence of various elements or values across the abstraction hierarchy, that any product must satisfy, while also being implementable by a subset of the reusable assets available in the SPL. The existence of this common core results from the main purpose of any VM: supporting semi- or fully-automated configuration of a particular product out of the product line. Typically, this configuration process is divided into two stages. The first consists of choosing one valid point in the problem space represented by alternatives in the VM. The second consists of deriving a working implementation solution from the reusable assets associated with the options selected during the first stage.

As SPLs grow larger, VMs grow increasingly complex. Real-life industrial SPLs routinely contain over 10K elements and constraints. Since the problem and solution spaces are subtly but sparsely constrained combinations of the optional and alternative VM elements, their sizes are subject to a combinatorial explosion. This makes fully manual configuration impractical. It also inevitably leads to the introduction of inconsistent elements or constraints during the engineering and evolution of the VM. Therefore, as with any software artifact, the VM needs to be verified and validated with the help of automation tools.

A wide variety of approaches have been proposed to implement SPL VM verification and VM-guided SPL configuration tools. Just like for SPL VM languages, there is also currently no accepted standard API for such tools, though the overwhelming majority of them share a key feature in common: they rely on some form of *logical* knowledge representation and automated reasoning. This allows them to reuse practically scalable inference engines developed over the last 50 years by two research communities, the formal software engineering methods community and the artificial intelligence community. Four main classes of such engines have been extensively proposed and evaluated to automate SPL VM verification and VM-guided configurations: ***SATisfiability (SAT)*** solvers and their ***Satisfiability Modulo Theories (SMT)*** successors, ***Constraint Satisfaction Problem (CSP)*** solvers, ***Logic Programming (LP)*** engines and their ***Constraint LP (CLP)*** successors and ***Description Logic (DL)*** engines and their semantic web successors. No member of these engine families is a silver bullet for all SPL VM verification or VM-guided SPL configuration problems. They have subtle differences in expressiveness and performance on different kinds of problems, even if expressed in the same DSL VM language.

In this paper we propose a novel, light weight approach to bridge the gap between, on the one hand, the existing diversity and lack of standardization in VM languages, and, on the other hand, the existing diversity of logical languages that have been proposed to provide VM languages with formal semantics and are accepted as input by various classes of inference engines. Our approach is based on two key ideas. The first is to use a lightweight, declarative, textual syntax to specify both the concrete and abstract syntax of a VM DSL. This textual syntax is encoded both as Python objects from the Pydantic library [5] and as JSON files in the Open API web service standard [6]. It can be seen as a more agile alternative to the traditional Model-Driven Engineering [7] based on diagrammatic models, meta-models, and meta-meta-models. It is the subject of another publication under preparation. The second key idea, which is the focus of the present paper, is the proposal of the ***Common Logic Interchange Format (CLIF)*** [8] standard from the ***International Organization for Standardization (ISO)***, originally put forward to support interoperability among logical inference engines, to represent the formal semantics of any VM DSL in a solver-agnostic fashion. It starts from realizing that the four main classes of logical languages listed above and commonly used for VM verification and VM-guided SPL configuration are all essentially sub-languages of CLIF in terms of their expressiveness. In addition, CLIF is also the language used to define the formal semantics of the fUML [9] standard, the formal core of the Unified Modeling Language (UML) [10]. Therefore, any model-driven SPLE approach using the UML to model assets, could leverage the mapping from VM models to CLIF to provide a uniform formal semantics for the whole SPL model comprising both the VM and the asset model.

The main contribution of this paper is to propose a first step towards a common formal semantics for SPL VM based on an ISO standard. We show, with a couple of illustrative examples, how the semantics of two very different SPL VM graphical languages, Extended Feature Models for static SPL VMs and Sawyer et al's [11] extension of the ***Knowledge Acquisition in autOmated Specification (KAOS)*** modeling language [12] for context-aware dynamic SPL VMs, can both be uniformly expressed in CLIF. We also describe the architecture of the VariaMos tool that validates the approach by allowing one to specify, in CLIF, the semantics of a SPL VM DSL and then automatically generate the CLIF formula to logically represent this semantics for a specific, graphically edited SPL VM. Given that CLIF is expressive enough to capture the restricted subsets of ***First Order Logic (FOL)*** accepted as input by most constraint solvers (which will be touched upon in the following section), this contribution will allow the subsequent use of a variety of inference engines that can be tailored to each VM language. We demonstrate our approach within an open-source tool called VariaMos [13] that allows its users to specify the concrete visual syntax, the abstract syntax and the formal semantics in agile, declarative, textual and uniform fashion as JSON files. The formal semantics JSON specification then serves to associate abstract syntax elements with CLIF elements formulas.

The rest of the paper is organized as follows: in Section II, we present an overview of the background and work related to our approach; in Section III, we present our proposal for the use of CLIF as the standard formal semantics for Variability Modeling; in Section IV, present our CLIF translation mechanism by example, by examining the translation of two different modeling languages; in Section V, we elaborate on our use of CLIF and the specific dialect we have chosen; in Section VI, we present the overall architecture of our prototype and its implementation; in Section VII, we cover the limitations of our approach and outline planned future work; and, finally, in Section VIII, we present our conclusions.

II. Background and Related Work

There have been many approaches to establishing formal semantics for variability modeling languages; these have generally always been defined in the context of performing automated analyses of the constructed models. Since the constructs for each language vary, the corresponding semantics have always been defined as a function of the expressiveness of each language. Some of the first exploratory works on this topic proposed the use of first-order logic to provide the semantics for ***Basic Feature Models (BFMs)*** [14] (the simplest and original type of VM), though they essentially remained within the propositional core of FOL and only needed first order constructs to encode their semantics into manually constructed Prolog programs. As BFMs evolved, so too did their semantics, and, in particular, Benavides et al. [15] provided a characterization of Cardinality-based feature models as Satisfiability [16], Binary Decision Diagram [17] and (Boolean) Constraint Satisfaction Problems [18], all falling into the purview of first order theories.

There exist many variability modeling languages that are used for SPLE and beyond. The models constructed with these languages aim to capture the variability relations that exist within a given domain with the aim of expressing the set of allowable combinations of domain elements in products. These domain elements are commonly modeled as “features” that encode an end-user-facing piece of functionality [1]. The relations among these features make explicit the design constraints imposed both by the domain itself and the technological choices involved. There has been a considerable amount of work regarding the automated analysis of these models during the past few decades [19], such as automated configuration of products or finding errors in the models. These efforts primarily focused on models of a particular type, that is, feature models, which were originally proposed in [20] and have been since extended with additional constructs that increase their expressivity. A considerable amount of alternative modeling languages, and even syntactic variations of the aforementioned variability models have been proposed, each aiming to improve upon the characteristics of these feature models to support more expressive models that better capture the nature of the domain.

It has been noted in the literature that (finite domain) constraint solving approaches are those best suited to handle the expressivity of features models extended with numerical and symbolic constructs as surveyed by Benavides et al. [21]. This survey highlights several other semantic approaches that have been proposed in the literature, like the use of Description Logic [22] originally proposed by Wang et al. [23]. That being said, the overwhelming majority of approaches fit squarely in the realm of classical predicate (first order) logic.

In addition, most of the works here cited, and cited in the above surveys [19] [21], demonstrate that the approaches, whenever constructed to support tooling, transform the variability models directly into the representations amenable for analysis by the underlying solver technologies. This makes these formalizations difficult to reuse, compare, debug and render them inflexible to changes in the input language. We therefore diverge from these approaches and aim to construct a representation that directly encodes first order formulas, i.e., Common Logic [8], and in particular its machine- and human-interpretable syntax, the Common Logic Interchange Format or CLIF.

III. CLIF as standard formal semantics for Variability Modeling

One of the main contributions of this article is the proposal of the Common Logic [8] standard as the ideal target representation of the logical semantics of variability models. In particular, we propose the use of a fully conformant subset of the Common Logic Interchange Format (CLIF) as the preferred notation towards which transformation procedures should aim to produce their results. While CLIF’s expressivity surpasses that of First Order Logic (FOL) through some additional constructs allowing for infinite expressions, we consider that there is only need to support the constructs effectively contained in FOL (c.f. Section 6.5 of [8] for a deeper justification and discussion as to why this is admissible and does not fundamentally limit our expressiveness). The justifications for choosing Common Logic (and CLIF in particular) are threefold: first, its capacity to be as expressive as FOL means that it easily represents all constructs that are handled by the most commonly used tools for analysis [19] [21], namely Constraint Logic Programming over finite domains [24], constraint programming [18], SAT solvers [16] and SMT [25] solvers, among others; second, its status as an international standard with a normative and fully defined representation format (CLIF) makes it easily interoperable with other systems and understandable by humans and machines alike; and, finally, the Lisp-like S-expression derived syntax make parsing and managing models represented in CLIF simple. In addition, this same structure facilitates the generation of these expressions from model elements.

There is an additional angle to consider as to why CLIF is particularly suitable for providing the semantics of models. Real world SPL projects go beyond domain VMs, and model the concrete software assets or artifacts that are to be used to assemble software products. UML models model the structure and behaviour of software systems, and are one possible type of asset model. As highlighted in the introduction, there have been ongoing efforts to provide formal semantics for UML models for automated execution and analysis, which have been defined in CLIF [9] for a subset of UML models. This opens the door to a possible avenue for investigating the logical integration between variability models and UML-derived asset models within a single analysis framework. CLIF has also found use in other domains, such as the basis for a large

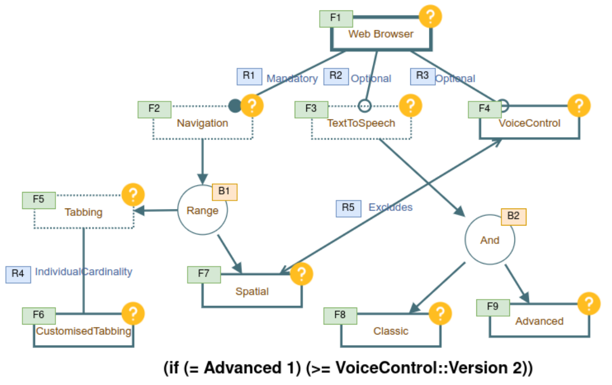


Fig. 1. An extended feature model with an arbitrary cross tree constraint depicted in VariaMos. Adapated from the example in Figure 2 in [29].

```
(model                                                      1
    (and (bool Web Browser) (= Web Browser 1))              2
    (bool Navigation)                                       3
    (bool TextToSpeech)                                     4
    (bool VoiceControl)                                     5
    (int VoiceControl::Version)                             6
    (bool Tabbing)                                          7
    (bool Spatial)                                          8
    (and (=< (Navigation * 1) (Tabbing + Spatial))          9
        (=< (Tabbing + Spatial) (Navigation * 2)) )         10
    (and (int (0 5) CustomisedTabbing) (and                 11
        (=< (Tabbing * 1) CustomisedTabbing)                12
        (=< CustomisedTabbing (Tabbing * 5)) ))             13
    (= (Classic + Advanced) (TextToSpeech * 2))             14
    (bool Classic)                                          15
    (bool Advanced)                                         16
    (= Web Browser Navigation)                              17
    (>= Web Browser TextToSpeech)                           18
    (>= Web Browser VoiceControl)                           19
    (=< (Spatial + VoiceControl) 1)                         20
    (if (= Advanced 1) (>= VoiceControl::Version 2))        21
)                                                           22
```

Fig. 2. The logical semantics of the feature model from Figure 1 in CLIF.

repository of formal ontologies [26], or as the input language for a tool that brings together heterogenous theorem provers [27].

IV. Illustrating VM language agnosticism by example

In this section, we aim to demonstrate the genericity of our approach by examining the translation of two different VM languages into CLIF. The key idea behind both of these examples is that, by providing each VM language with a specification for its semantics, we can transform any model constructed with said language into its corresponding logical semantics. These semantics could, in turn, be used as the input for inference engines with which different analyses could be performed. To achieve this, we make use of JSON [28] specifications that act as sets of templates for each of the elements present in each language's abstract syntax. These templates generate logic formulas with CLIF syntax, and, when collected together (with an implicit conjunction of all these formulas), form a complete CLIF model, that acts as the logical theory one associates to a given model.

To capture as large a gammut as possible of VM languages, we allow the semantics to be defined for all syntactic constructs that can be depicted in our modeling tool. In addition, some languages include constructs that reify, for instance, one-to-many relations that, to render their semantics, need information from neighboring nodes in the graph; therefore, we explicitly allow translation rules to capture information about neighboring elements in the graph to generate the CLIF formulas.

A. *Feature Models*

Figure 1 depicts an extended feature model for a product line of accesible web browsers featuring cardinalities and attributes. All the elements of the model have been annotated and numbered according to their type: features are in green; relations in blue; and elements that reify one to many relations (with UML-like cardinality ranges), called bundles, are in light orange. The logical semantics of the model in CLIF are portrayed in Figure 2. The correspondence between the model and its semantics is as follows:

- Line 2 depicts the semantics of F1 as a boolean variable, and, since it is a root feature, it also models a constraint that obligates it to be present, i.e., set to 1.
- Lines 3–5, 7–8, and 15–16 represent features F2–5 and F7–9 as boolean variables.
- Line 6 represents the Version attribute (not visible in the figure) of F4 as an integer variable.
- Line 9 represents the semantics of the reified relation B1, giving a range of 1 to 2 selected features if feature F2 is present.
- Line 11 represents the semantics of F6, a feature that, unlike the others, is not boolean, but can instead be present as up to 5 instances (also called clones in the literature).
- Line 14 represents the reified "and" relation B2, implying that if the parent feature F3 is present both F8 and F9 must be present aswell.
- Line 17 represents the mandatory relation R1, i.e., the two features must be bound to the same value.
- Lines 18 and 19 represent the optional relations from F1 to F3 and F4.
- Line 20 encodes the exclusion relation between F7 and F4.
- Line 21 encodes a complex constraint between F9 and the Version attribute of F4.

The transformation of the model and its elements is done through the specification of the semantics of the VM language's syntax elements in a JSON format. This JSON serves to provide a set of "templates" to turn these abstract syntax elements of the model into CLIF expressions. Figure 3 presents a fragment of the translation rules necessary to transform a feature model into its corresponding CLIF model. These templates form a bridge between the graph structure of the model and the CLIF expressions that represent them:

- Lines 1–7 define the semantics of the (boolean) features

```
1  {
2    "elementTypes": ["ConcreteFeature", ...],
3    "elementTranslationRules": {
4      "ConcreteFeature": {
5        "param": "F", "constraint": "(bool F)", ...
6      }
7    },
8    "relationTypes": ["Excludes", ...],
9    "relationPropertySchema": {
10     "type": { "index": 0, "key": "value" }
11   },
12   "relationTranslationRules": { ...,
13     "Excludes": { "params": ["FA","FB"],
14       "constraint": "(=< (F1 + F2) 1)"
15     },
16     "Optional": { "params": ["FA","FB"],
17       "constraint": "(>= F1 F2)"
18     },
19     "Mandatory": { "params": ["F1","F2"],
20       "constraint": "(= F1 F2)"
21     }
22   },
23   "relationReificationTypes": ["Bundle"],
24   "relationReificationTranslationRules":{
25     "Bundle": { "param": ["F","Xs","min","max"],
26       "paramMapping": {
27         "inboundEdges": {"unique": true,"var": "F"},
28         "outboundEdges": {"unique": false,"var": "Xs"}
29       },
30       "constraint": { ..., "And": "(= (sum(Xs)) (F * len(Xs)))",
31         "Range": "(and (=< (F*min) (sum(Xs))) (=< (sum(Xs)) (F*max)))"
32       }
33     }
34   },
35   ...
36 }
```

Fig. 3. Fragment of Feature Model Semantic Translation specification JSON.

as boolean variables where where *bool* is a distinguished unary predicate defining the domain of the variable F, i.e., $F \in \{0, 1\}$.

- Lines 8–21 define the semantics of the relations between features as CLIF expressions with arithmetic predicates and the mechanism for determining their type according to their properties.
- Lines 23–34 define the semantics of the bundles, and, given their one-to-many nature, define which of the two sides (ingoing or outgoing) contains multiple edges to expand expressions like *sum* or use their length in the CLIF expressions. In addition, depending on the type of the bundle, additional properties of the node may play a role, like the min/max properties for a range.

B. *Sawyer et al.'s Variability Modeling Langauge*

Figure 4 depicts a fragment of the principal model, using a modified KAOS [12] language, proposed in [11]. It models a flood early-warning system and how the system can modify its operational configuration depending on the state of its environment as reported through sensors. The language they have defined is structured as follows:

- Goals, with labels in green, determine the functional requirements of the system and are analogous to features in feature models. Goals form a hierarchy wherein the lower level goals imply how the goals they point to are to be achieved.
- Soft Goals, depicted as clouds annotated in Blue, encode the non-functional requirements of the system and can be satisfied in a 0 to 4 scale, which is encoded as "--", "-", "=", "+", "++" in the model. They themselves form a hierarchy in a manner analogous to goals.
- Context Variables, annotated in light red, encode the state of the system's environment among a set possible choices enconded as an enumeration of strings.
- Operationalizations, labeled in gray, specify the ways in which a goal can be satisfied and correspond to concrete modes of operation of the system. They are tied to a goal through Bundles, in light orange, which behave analogously to feature models (though with the edge direction reversed).
- Claims, annotated in magenta, express the level to which operationalizations satisfy the Soft Goals as a function of which has been selected.
- Soft Influences, labeled in yellow, relate the context variables to the Soft Goals, and determine the required level of satisfaction when the given state is determined by the context, e.g., if CV1 is "Low", the required level of satisfaction of SG5 is "++".

The aim with this language is to construct an optimization problem where the largest amount of claims can be satisfied in terms of the selected operationalizations, and therefore ensure that the Soft Goals are satisfied to a given level.

As before, we can interpret the semantics interpretation of the CLIF model in Figure 5 as follows:

- Lines 2–4, 11–14 encode the the Goals G1–3 and the operationalizations O1–4 as boolean variables. In addition, lines 45 and 46 encode the relations of the subgoals to the main goal.
- Lines 5–10 encode the value relations of the Softgoals with those above them in the hierarchy as their average.
- Lines 15–18 encode the bundles B1–2 as the choices between the operationalizations.
- Lines 19–30 encode the consequences of the claims C1–4 on the Soft Goals, with the claims themselves being a boolean variable that is true iff their claims are satisfied in the resulting configuration.
- Lines 31–32, 35–40 encode the semantics of the Soft Influences on the Soft Goals, and, just like the claims, are boolean values that are true iff the requirements are satisfied.
- Lines 33–34 encode the Context Variables CV1–2 as enumerations in a fixed range.
- Lines 41–44 encode the Soft Goals as bounded integer variables.

In Section V, we will continue with the analysis of this example, its semantic specification and the implications for representing VMs in CLIF.

V. Differences between the dialect used for semantic specification and (standard) CLIF

As was hinted at in the previous section, while we target CLIF as our representation, there are some practical consid-

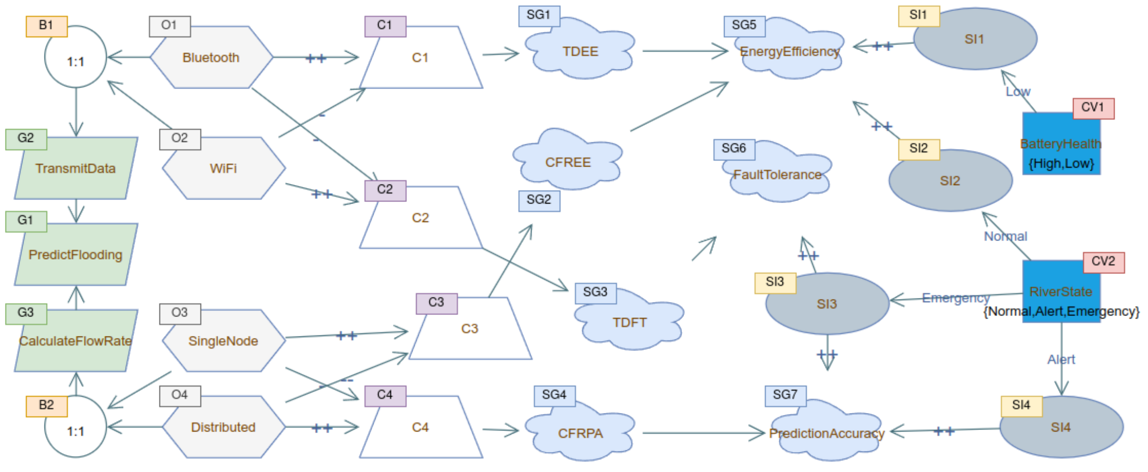


Fig. 4. A fragment of the model for the run-time variability of a flood early-warning originally proposed in and using the language of Sawyer et al.'s [11].

```
(model
    (bool PredictFlooding)
    (bool TransmitData)
    (bool CalculateFlowRate)
    (and (int (0 4) EnergyEfficiency)
        (= EnergyEfficiency ((CFREE + TDEE )/2)))
    (and (int (0 4) FaultTolerance)
        (= FaultTolerance ((TDFT)/1)))
    (and (int (0 4) PredictionAccuracy)
        (= PredictionAccuracy ((CFRPA)/1)))
    (bool Bluetooth)
    (bool WiFi)
    (bool Distributed)
    (bool SingleNode)
    (and (=< (TransmitData * 1) (Bluetooth + WiFi))
        (=< (Bluetooth + WiFi) (TransmitData * 1)))
    (and (=< (CalculateFlowRate * 1) (SingleNode + Distributed))
        (=< (SingleNode + Distributed) (CalculateFlowRate * 1)))
    (and (bool C1) (iff (= C1 1) (and
        (if (= Bluetooth 1) (=< TDEE 4))
        (if (= WiFi 1) (=< TDEE 1)))))
    (and (bool C2) (iff (= C2 1) (and
        (if (= Bluetooth 1) (=< TDFT 1))
        (if (= WiFi 1) (=< TDFT 4) ))))
    (and (bool C3) (iff (= C3 1) (and
        (if (= SingleNode 1) (=< CFREE 4))
        (if (= Distributed 1) (=< CFREE 0)))))
    (and (bool C4) (iff (= C4 1) (and
        (if (= SingleNode 1) (=< CFRPA 1))
        (if (= Distributed 1) (=< CFRPA 4)))))
    (and (bool SI1) (iff (= SI1 1) (if (= BatteryHealth 0)
        (and (= EnergyEfficiency 4)))))
    (enum (0 1 2) BatteryHealth)
    (enum (0 1) RiverState)
    (and (bool SI2) (iff (= SI2 1)
        (if (= RiverState 0) (and (= EnergyEfficiency 4)))))
    (and (bool SI3) (iff (= SI3 1) (if (= RiverState 2)
        (and (= FaultTolerance 4) (= PredictionAccuracy 4)) ) ) )
    (and (bool SI4) (iff (= SI4 1)
        (if (= RiverState 1) (and (= PredictionAccuracy 4)))))
    (int (0 4) TDEE)
    (int (0 4) CFREE)
    (int (0 4) TDFT)
    (int (0 4) CFRPA)
    (= TransmitData PredictFlooding)
    (= CalculateFlowRate PredictFlooding)
)
```

Fig. 5. The logical semantics of the feature model from Figure 1 in CLIF.

erations for its use that mean that we differ from CLIF as presented in the standard. These deviations, though small, have important consequences for the models produced through the semantic specification mechanism. The first of these departures is that we expressly recognize a set of distinguished predicates beyond the sole equality recognized by CLIF, such as unary or binary predicates like *int and bool* relating to the domains of variables used in the program. This is motivated by a desire to facilitate the construction of executable representations in different solvers that generally cannot automatically infer domain membership or require it outright for every variable. The other quite salient departure from the standard presentation of CLIF relates to the treatment of quantifiers; in effect, to enable complex arbitrary constraints, or even merely support the conjunction of the sentences outlined in the example from subsection IV-B, the reocurrence of the same variables must imply that they refer to the same object. Therefore, variables that occur free in the generated semantics are to be interpreted as being implicitly universally quantified over the conjunction of all the generated sentences; we do this following the tradition set by the standard first-order semantics for (pure) Prolog programs given by Clark's completion [30] and that of the ***Knowledge Interchange Format*** [31] language from which CLIF itself was derived. For example, the full logical reading of the exclusion relation in Figure 1 and therefore lines 5,8 and 20 from Figure 2 would be (using their annotated names for brevity):

$$\forall F_4, F_7 \bigwedge\{(F_4 \in \{0,1\}), (F_7 \in \{0,1\}), (F_4 + F_7 \leq 1)\}$$

This naturally leads to the question of handling quantifiers within each of the sentences. We have not yet found it necessary to introduce existential quantifiers for the semantics of the models languages we have dealt with so far; however, we have found very important applications of universal quantifiers, in particular, as a construct allowing one to deal with sentences that involve sets of incoming or outgoing edges to a given node in the directed graph. We currently consider, then, that the quantifiers range over finite sets that are defined precisely by the multiplicity of possible connections. This, in turn, leads to an important transformation that can be done directly without any loss of expressivity, namely that one can transform the universal quantification over a single variable and over a sentence into the iterated conjunction of a set of sentences each being the original sentence with the bound variable replaced by a member of the set. This is due to the following equivalence:

Let S be a finite set and ϕ an arbitrary first order sentence with only s unbound.

$$\forall s \in S \phi(s) \Leftrightarrow \bigwedge_{s \in S} \phi(s)$$

This greatly simplifies, and augments the power of, the semantics we can express and in turn greatly simplifies the translation from CLIF towards some solver paradigms where there is no pure or declarative handling of classical universal quantification, e.g., Prolog. To be clear, the quantifier expansion draws its values from the graph's elements and not from the variable domains over which we are solving. This subtle point means that we retain the intensive expression of the semantics but are able to write more general rules that have potentially varying amounts of appearances of certain elements from neighboring elements.

To demonstrate these mechanisms and differences in practice, consider the example from Figure 4, and in particular centered on the element SI3 and its relation with its neighbors SG6, SG7 and CV2. We have defined the semantics of the Soft Influence as shown in Figure 6. Within this semantic specification, we "bind" the set over which the x variable is quantified as the target nodes of the outbound edges, corresponding to X_s in the translation rule and to SG_6 and SG_7 in Figure 4. We also have distinguished functions relating to intrinsic properties of the directed graph, such as $edge(x)$ whose value is the edge leading to the x node, and we also allow references to arbitrary custom properties defined on graph elements through the :: operator.

The corresponding logical reading of these semantics would be as follows:

Let X be the set of nodes in the graph, $X_s \subset X$ be the set of nodes corresponding to the outgoing edges in the model, E be the set of edges in the model, $edge$ be a function $edge : X \to E$, V the set of attribute types and V_s the set of attribute values, :: be an infix binary function $(::) : E \cup X \times V \to V_s$, S be the variable corresponding to the id of the soft influence node, and $F \in \{0, 1\}$ the variable corresponding to the id of the unique inbound node. It is to be understood that the predicate imposing bounds on the F variable would also be part of the semantics.

$$\forall S, F \big[bool(S) \wedge \big((S = 1) \iff (F = edge(F) :: Value \\ \implies \forall x \in X_s (x = edge(x) :: SatisfactionLevel)) \big) \big]$$

However, given the equivalence cited above, the rendered semantics would be (lines 37–38):

```
{ ...,
  "relationReificationTranslationRules": {
    "SoftInfluence": {
      "param": ["S", "F", "Xs"],
      "paramMapping": {
        "node": "S",
        "inboundEdges": { "unique": true, "var": "F" },
        "outboundEdges": { "unique": false, "var": "Xs" }
      },
      "constraint": {
        "SoftInfluence": "\
        (and (bool S) (iff \
          (= S 1) (if \
            (= F edge(F)::Value)
            (forall (x:Xs) \
              (= x edge(x)::SatisfactionLevel) \
            ) \
          ) \
        ) )"
      }
    }, ...
  }, ...
}
```

Fig. 6. Fragment of Sawyer et al.'s language's Semantic Translation specification JSON. The "\" indicates the split in the multiline string for readability and formatting.

```
(and (bool SI3) (iff (= SI3 1)
  (if (= CV2 edge(CV2)::Value) (and
    (= SG6 edge(SG6)::Value) (= SG7 edge(SG7)::Value)))))
```

This removes the need to handle the internal quantifier, and simplifies the reading of the rendered formula.

VI. Variability Model to CLIF Translation Architecture and Implementation

We propose a distributed architecture for CLIF semantic translation, as depicted in Figure 7. The necessary tooling for modeling is served from a cloud infrastructure avoiding the need to perform any installation on the client beyond browsing to the website, where the user is served the user interface (shown in green) by the FrontEnd HTTP server. Within the cloud infrastructure, all concerns relating to the storage of the languages are handled, including their syntax and semantics, with a database backing these operations and providing a source of persistence between client interactions. An additional service that can perform certain checks on graph validity, such as allowed element amounts and connections beyond what can be handled natively by the graphics library used by the client is proposed in the architecture, to offload some responsibilities from the client. All of these services are deployed as containers, and are shown in white in our logical architecture diagram.

When it comes to the semantic translator service (shown in yellow), which is the core tool covered by this articles, it has been developed as a Docker [32] container which will expose a REST API endpoint over which the translation (and eventual analysis) operations will be served. Since our Front End is inherently configurable, this Back End can be deployed anywhere. For our prototype we have deployed it locally within a test cluster, but it will be made available within the

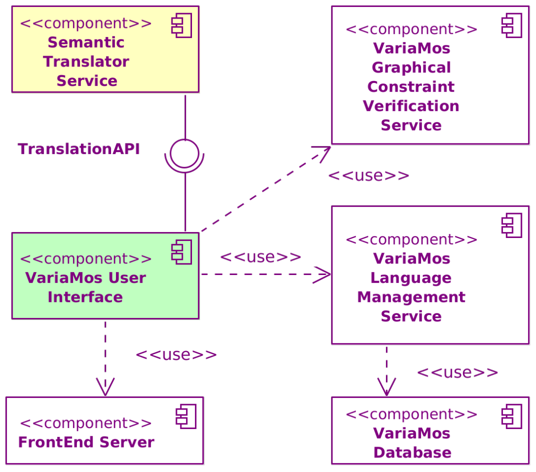


Fig. 7. High Level logical architecture of the VariaMos tool and the translation tool.

VariaMos cloud infrastructure, where it will be the default point of access for translation. Looking forward, however, to the integration of the underlying solvers (which may also require licenses and hence can only be run locally), it seems clear that decoupling the translation mechanism allows for the best use of the available computational resources by allowing the user to choose where he desires the operations to be run, either sharing a cloud resource with others or deploying a local version of the container if he so wishes. This retains all the benefits of a cloud-native solution, while also being flexible when additional computational resources are required for a particular project.

The high-level operating principles of the translation mechanism are described in Figure 8. The fundamental operations carried out involve performing data validation on both the form of the provided Model and the semantics. Then the serialized model is reconstructed into a Graph; this, put together with the semantics, are then put through the CLIF model generation procedure which ultimately outputs the model's semantics which are the reported back to the user. This is all exposed through the API endpoint served in the translator container.

In terms of the implementation of the translation Back End, we have constructed the server in python with the following software components: Flask [33] for the server code, pydantic [5] for data schema validation, networkX [34] for graph representation in python, and textX [35] for managing the CLIF grammar. All of the code for the translator is open-source and freely available on GitHub at [36]. Within this repository are included the full grammar for our subset of CLIF, and complete examples (including models, semantics and syntax) for basic and extended feature models, as well as for Sawyer et al.'s [11] modified version of KAOS.

VII. Limitations and Future Work

In terms of our approach's limitations, it must be noted that we do not cover the entire CLIF standard, and we have not yet found a need to construct a particular treatment of existential

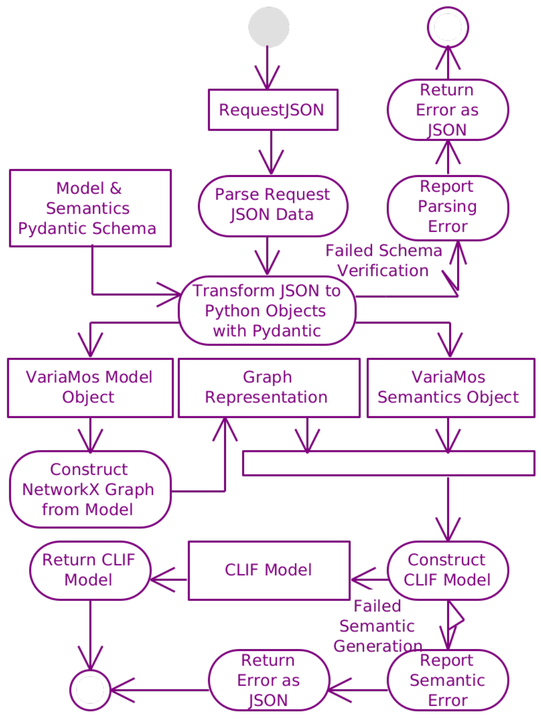


Fig. 8. High-Level operation of the translation tool.

quantification within model's semantics. There are also some limitations tied to the translation mechanism for the generation of the semantics, namely, we require that the elements that participate in a given node's or edge's semantics be directly connected, and thus we cannot yet perform the transitive traversal of the graph for the generation of the semantics. We also have no semantic treatment for elements that can be nested. We also require the user to have some understanding of the internal structure of the graph representation in the front-end client in order to define rules for attribute lookup to, for example, perform type disambiguation when a given node or relation's semantics depends on types defined by these attributes. This means that while we believe our approach is expressive enough to handle most VM languages that have been covered in the literature, it is possible that some others posses constructs that aren't easily expressible or require the aspects of CLIF we have not yet covered, such as deeply nested negation. In addition, we are restricted to languages that form directed graphs, so we are unable to treat languages with more complex graph types like hypergraphs without some measure of reification. We also do not handle the translation of textual languages into CLIF.

As mentioned in Section VI, our principal aim is to continue expanding upon the already implemented aspects of the proposal in order to complete the full end-to-end cycle of model generation and subsequent automated analysis through the use of several solvers. We will initially target constraint solvers as these are the best attested in the literature [37] and have the most straightforward translation from CLIF into their respective representation, but we ultimately aim to support a larger range of first order logic-based automated analysis tools.

VIII. Conclusions

In this article, we have presented a proposal for the use of CLIF as the standard representation format for the semantics of Variability models. We also present and demonstrate a mechanism to specify formal semantics for variability modeling languages through a simple JSON based specification format.

Our mechanism leverages the user-friendliness of the JSON format with the ability to quickly construct one's needed (and formally defined) modeling language semantics in such a manner that it spares prospective users from the need to learn the specifics of the programming involved. It also enables the quick evolution of language's semantics with no modifications to the underlying tools.

We believe this will permit the transparent integration of multiple analysis methods and especially solver families through the construction of translation from CLIF into their respective syntaxes.

Acknowledgment

We would like to thank Mauricio Agudelo for his work on the architecture and development of VariaMos, Jairo Soto for his work on the deployment of the VariaMos cloud infrastructure, and Francisco Piedrahita for his work on an initial version of these ideas.

References

[1] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*, 1st ed. New York, NY: Springer, 2005.

[2] Systems and Software Product Line Conference, "Product Line Hall Of Fame," https://splc.net/fame.html, n.d., accessed: 2023-03-27.

[3] B. H. Cheng *et al.*, "Using models at runtime to address assurance for self-adaptive systems," *Models@ run. time: foundations, applications, and roadmaps*, pp. 101–136, 2014.

[4] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *Computer*, vol. 41, no. 4, pp. 93–95, 2008.

[5] Pydantic Services Inc., "Pydantic," https://pydantic.dev/, 2023, accessed: 2023-03-27.

[6] OpenAPI Initiative, "OpenAPI Specification," https://spec.openapis.org/oas/latest.html, 2021, accessed: 2023-03-27.

[7] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*, ser. Synthesis lectures on software engineering. Morgan and Claypool, 2017.

[8] "Information Technology – Common Logic (CL) – A framework for a family of logic-based languages," International Organization for Standardization, Geneva, CH, Tech. Rep., Jul. 2018.

[9] "Semantics of a Foundational Subset for Executable UML Models (fUML), version 1.5," Object Management Group, Tech. Rep., Apr. 2021.

[10] S. Cook *et al.*, "Unified Modeling Language (UML), version 2.5.1," Object Management Group, Tech. Rep., Dec. 2017.

[11] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes, "Using constraint programming to manage configurations in self-adaptive systems," *Computer*, vol. 45, no. 10, pp. 56–63, 2012.

[12] A. Van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software*. John Wiley & Sons, 2009.

[13] VariaMos Team, "VariaMos Framework," https://variamos.com/, 2023, accessed: 2023-03-27.

[14] M. Mannion, "Using first-order logic for product line model validation," in *Software Product Lines: Second International Conference, SPLC 2 San Diego, CA, USA, August 19–22, 2002 Proceedings*. Springer, 2002, pp. 176–187.

[15] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, "A first step towards a framework for the automated analysis of feature models," *Proc. Managing Variability for Software Product Lines: Working With Variability Mechanisms*, pp. 39–47, 2006.

[16] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," *Foundations of Artificial Intelligence*, vol. 3, pp. 89–134, 2008.

[17] R. Drechsler and D. Sieling, "Binary decision diagrams in theory and practice," *International Journal on Software Tools for Technology Transfer*, vol. 3, no. 2, pp. 112–136, May 2001.

[18] R. Dechter and D. Cohen, *Constraint Processing*. Morgan Kaufmann, 2003.

[19] D. Benavides, "Variability Modelling and Analysis During 30 Years," in *From Software Engineering to Formal Methods and Tools, and Back: Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday*, ser. Lecture Notes in Computer Science, M. H. ter Beek, A. Fantechi, and L. Semini, Eds. Cham: Springer International Publishing, 2019, pp. 365–373.

[20] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study:," Defense Technical Information Center, Fort Belvoir, VA, Tech. Rep., Nov. 1990.

[21] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, Sep. 2010.

[22] F. Baader, D. Calvanese, D. McGuinness, P. Patel-Schneider, and D. Nardi, *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge university press, 2003.

[23] H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan, "A semantic web approach to feature modeling and verification," in *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, 2005, p. 46.

[24] J. Jaffar and J.-L. Lassez, "Constraint logic programming," in *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1987, pp. 111–119.

[25] C. Barrett and C. Tinelli, "Satisfiability Modulo Theories," in *Handbook of Model Checking*, E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, Eds. Cham: Springer International Publishing, 2018, pp. 305–343.

[26] Semantic Technologies Laboratory, "COLORE," http://stl.mie.utoronto.ca/colore/, n.d., accessed: 2023-03-27.

[27] T. Mossakowski, M. Codescu, O. Kutz, C. Lange, and M. Grüninger, "Proof support for common logic." in *ARQNL@ IJCAR*, 2014, pp. 42–58.

[28] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," Internet Engineering Task Force, Request for Comments RFC 4627, Jul. 2006.

[29] J. Carbonnel, M. Huchard, and C. Nebut, "Towards complex product line variability modelling: Mining relationships from non-boolean descriptions," *Journal of Systems and Software*, vol. 156, pp. 341–360, Oct. 2019.

[30] J. W. Lloyd and J. W. Lloyd, *Foundations of Logic Programming*, 2nd ed., ser. Artificial Intelligence. Berlin Heidelberg: Springer, 1993.

[31] M. R. Genesereth and R. E. Fikes, "Knowledge interchange format-version 3.0: Reference manual," 1992.

[32] Docker Inc., "Docker Documentation," https://docs.docker.com/, 2023, accessed: 2023-03-27.

[33] The Pallets Projects, "Flask User's Guide," https://flask.palletsprojects.com/en/2.2.x/, n.d., accessed: 2023-03-27.

[34] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[35] I. Dejanović, R. Vaderna, G. Milosavljević, and Ž. Vuković, "TextX: A Python tool for Domain-Specific Languages implementation," *Knowledge-Based Systems*, vol. 115, pp. 1–4, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705116304178

[36] C. Correa Restrepo, "Semantic Translator," https://github.com/ccr185/semantic_translator, 2023, accessed: 2023-03-27.

[37] M. Pol'la, A. Buccella, and A. Cechich, "Analysis of variability models: A systematic literature review," *Software and Systems Modeling*, vol. 20, no. 4, pp. 1043–1077, Aug. 2021.

A Federated Source Code Quality Query and Analysis Platform

Tugkan Tuglular
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: tugkantuglular@iyte.edu.tr

Onur Leblebici
Univera Inc.
Izmir, Turkiye
email: Onur.Leblebici@univera.com.tr

Emre Baran Karaca
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: emrekaraca@std.iyte.edu.tr

Naşit Uygun
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: nasituygun@std.iyte.edu.tr

Osman Anıl Hiçyılmaz
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
email: osmanhicyilmaz@std.iyte.edu.tr

***Abstract*—The typical approach to data analysis is to store, query, and analyze data in a central location. In the case of source code, where multiple organizations or partners in a consortium contribute to a software, the repositories would be distributed and might be private. Within such a setting, one goal would be achieving and maintaining a certain level of source code quality across the consortium. One solution is to consider each partner as a node in a federated network. This paper proposes a federated code quality query and analysis platform. It further presents the features and the design of this platform.**

Keywords-source code quality; federated network; federated query; federated analysis.

I. INTRODUCTION

There are cases where each partner in a consortium, such as in the NESSI-SOFT project [1] in the Sixth Framework Programme and in the MODUS project [2] in the Seventh Framework Programme, does not want to share all of its source code but needs to be queried whether holding a pre-determined minimum source code quality level so that a certain level across the consortium is achieved and maintained. For such cases, one solution is to build a federated network so that each node in this network has its privacy, but shares required quality information. This paper considers this setting for source code quality and proposes a Federated Source Code Quality Query and Analysis (FSCQQA) platform. The setting is visualized in Figure 1.

The FSCQQA platform consists of a central site as seen at the top of Figure 1 and multiple sites, which are peers. It is a kind of peer-to-peer network, where the peers accept and follow a general policy and corresponding rules. In addition, the central site is responsible for inclusion and removal of peer sites with respect to the general policy. Such platforms are on the rise especially in the health field, where privacy regulations and expectations are high, and accountability is enforced at state level. The proposed FSCQQA platform is one of the early attempts, where the idea is applied to source code, but not health records. Therefore, we believe that there is a practical gain from such a platform proposal.

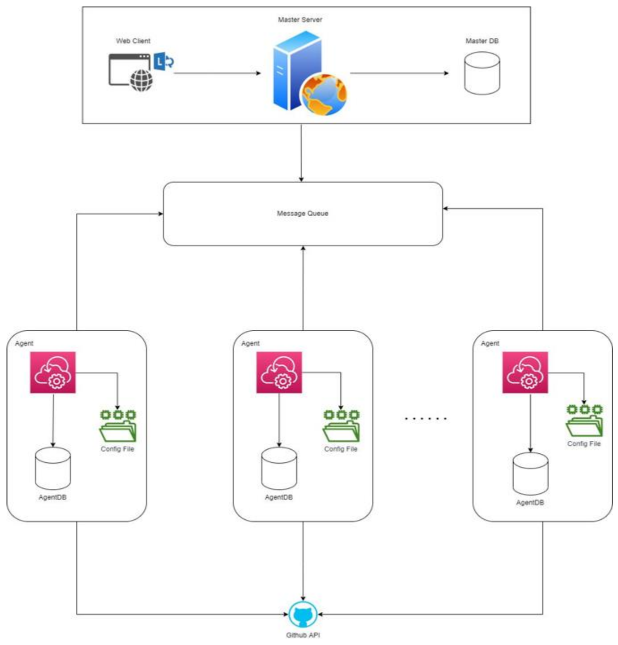


Figure 1. The FSCQQA platform overview.

The proposed platform is not only for consortiums to utilize. A global software company with development sites in various countries can also benefit from the FSCQQA platform. In this setting, concerns like revealing too much information about the software under development and the software development team may be relieved.

The FSCQQA platform offers opportunities for querying and monitoring source code quality across a consortium. This platform can facilitate analyzing how source code improvements are performed and how defect numbers are minimized. The FSCQQA platform has the following features:

- Analyze software quality with defect and source code metrics.
- Share defect and source code metrics with peers and consortium administration/management.
- Follow trends and improve.
- Compile federated historical data on defects and source code quality.

The features are kept at minimum in the paper, but they can be extended easily. To serve these features, the FSCQQA platform provides a data infrastructure, a software stack, and the operations on them. The proposed design is novel. The FSCQQA platform can be used for source code quality and defect prediction in the future.

As of today, there are multi-site software development companies whose sites are globally distributed. Each site is autonomous to some degree, but they are subject to central management rules. In such a setting, tracking each site's software quality and achieving an overall performance is not easy. Such a platform would be beneficial to them as well.

The paper is organized as follows: Section II presents the bug, or defect, datasets and source code quality metrics. Section III explains the proposed platform. Section IV outlines related work, and the last section concludes the paper.

II. FUNDAMENTALS

A. Bug Datasets

Lately, bug datasets are composed for bug or defect prediction. Following this, Ferenc et al. [3] compiled and standardized existing public bug datasets. The same group [4] extended their bug dataset and made the dataset publicly available at [5]. Several research works have produced and utilized bug datasets to develop and evaluate novel bug prediction methods. The objective of their study is to collect and combine current public source code metrics-based bug databases. In addition, they evaluated the abundance of gathered metrics and the bug prediction skills of the unified bug dataset. One research direction in this field moves toward combining bug datasets with software code quality metrics for better prediction. One of the first attempts is published by Osman et al. [6]. They evaluated sixty distinct bug prediction setting combinations on five open-source Java projects using a cost-aware evaluation scheme. Change measurements combined with source code metrics were discovered to be the most cost-effective option for developing a bug predictor. Another example of this work is presented by Mashhadi et al. [7]. They conducted a quantitative and qualitative study on two prominent datasets (Defects4J and Bugs.jar) utilizing 10 common source code metrics, as well as two popular static analysis tools (SpotBugs and Infer), for the purpose of evaluating their capacity to anticipate flaws and their severity.

B. Source Code Quality Metrics

Software quality metrics have been proposed for decades. The literature starts in 1970s. In the 1980s and 1990s, design metrics and their impact on software and source code were mainly studied. Henry and Selig [8] published a book on design metrics, which predicts source code quality. Two early research works specifically on source code quality metrics are by Pearse and Oman [9] and by Welker et al. [10]. They worked on the maintainability of source code.

With the popularity of object-orientation, the research in this area was intensified. Nuñez-Varela et al. [11] did a comprehensive mapping investigation on 226 articles that were published between 2010 and 2015 and discovered nearly 300 source code metrics. Even though object-oriented metrics have received a great deal of attention, there is a need for greater research on aspect and feature-oriented measurements. Prediction of software faults, complexity, and quality evaluation were recurring themes in these investigations.

Currently, there are separate tools as well as tools embedded into platforms, which not only produce source code quality metrics but also calculate technical debt. The next step for these tools seems to be towards predictions and suggestions for better code quality. Our vision and current attempt are in the same direction.

III. PROPOSED PLATFORM

We propose a federated code quality query and analysis platform, called FSCQQA. In this section, we first explain our design goals, such as "authentication and authorization" and "logging and monitoring" and continue with the services the FSCQQA platform provides. Some local services may vary between sites, but standardized procedures and rules will be implemented to ensure uniform administration and oversight. Finally, we present our user interface design to give a sense of use cases for the FSCQQA platform.

A. Design Goals

The major design goals are as follows:

Authentication & Authorization (AA): Each partner or site may have its own AA mechanism implemented. Then, each partner is responsible for the FSCQQA platform for its users' queries. Each query includes the user and site identification; the site is responsible for logging the queries.

Access Control (AC) Policies: Each site may have its policies and regulations depending on the country where the site is. Therefore, the response to each query is filtered locally before sending. Each site should guarantee that any response does not contain any personal identifiable information.

Secure Communication: Each site must be able to communicate securely with trustworthy peers. All nodes exchange secure Public Key Infrastructure certificates in order to establish trust. While the FSCQQA platform is a federated network, the security of the nodes is only as strong as the network's weakest link.

Logging and Monitoring: Every query executed by a node should be recorded in an audit trail that the peer sites could view. The logs will be monitored by the central site for anomalies.

Standard APIs: Each site should provide standard APIs defined by the FSCQQA platform. Although the FSCQQA platform provides a software agent called FSCQQA agent to fulfil this requirement, the site may choose to implement its own software agent.

Source Code Repositories: The FSCQQA platform provides a software agent to work with GitHub [12] repositories. However, this is not a must. Any site can work with any source code repository but must ensure that standard APIs required by the platform are provided.

Management of Federated Platform: There is a central site responsible for the management of partners and their sites. These management operations include adding and removing partners and sites (a partner may have more than one site), constantly informing partners about other alive partners and sites, and collecting velocity and trend information from site.

B. Services

The FSCQQA platform defines two types of services, one provided by the FSCQQA agent and the other by the standard FSCQQA APIs. The FSCQQA agent is customizable through configurations with the following parameters:

- GitHub repository address
- GitHub repository access rights

The FSCQQA agent automatically generates local defect database for each site from a GitHub repository by extracting commit/issue histories and analyzing them. At the same time, it collects software metrics, such as lines of code and cyclomatic complexity, for each commit/issue. The defect information with software metrics will represent source code quality of the software developed at a site. Moreover, the FSCQQA agent extracts source code related metrics for a specific version using tools, such as OpenStaticAnalyzer [13]. The process is presented as an Unified Modeling Language (UML) sequence diagram in Figure 2. The FSCQQA agent is also responsible for the management of the local database for defects and metrics. To mitigate security concerns related to such an agent software, its source code should be open.

The standard FSCQQA APIs provide the services of the FSCQQA platform with respect to Open-API specifications [14]. The services are grouped as follows:

- Defect related metrics: number of existing (active) defects, defect density, defect resolve velocity, longest unresolved defect.
- Source code related metrics: class metrics, method metrics, coupling metrics, cohesion metrics, cyclomatic complexity metrics.

The services provide data for a specific version. They can be extended to supply data between two versions, but it may complicate the presentation of information and is, therefore, left as future work. The service calls can be for a specific metric or a set of metrics from a specific site or the whole network. If the whole network is queried, the query site requests all alive sites from the central site and queries each one individually then accumulates the results.

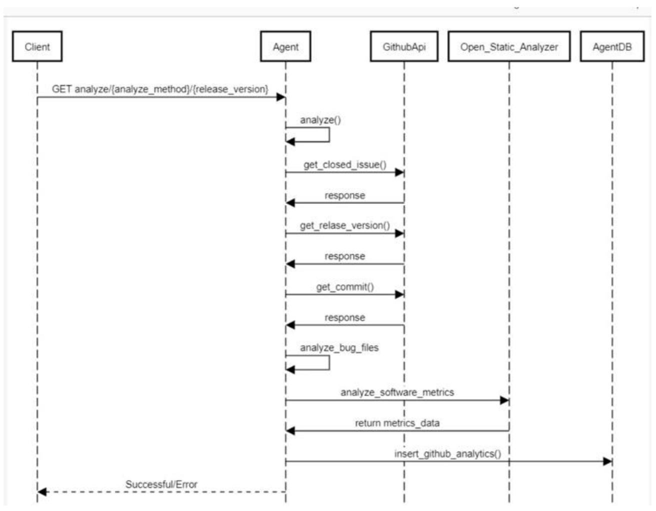


Figure 2. The FSCQQA platform overview.

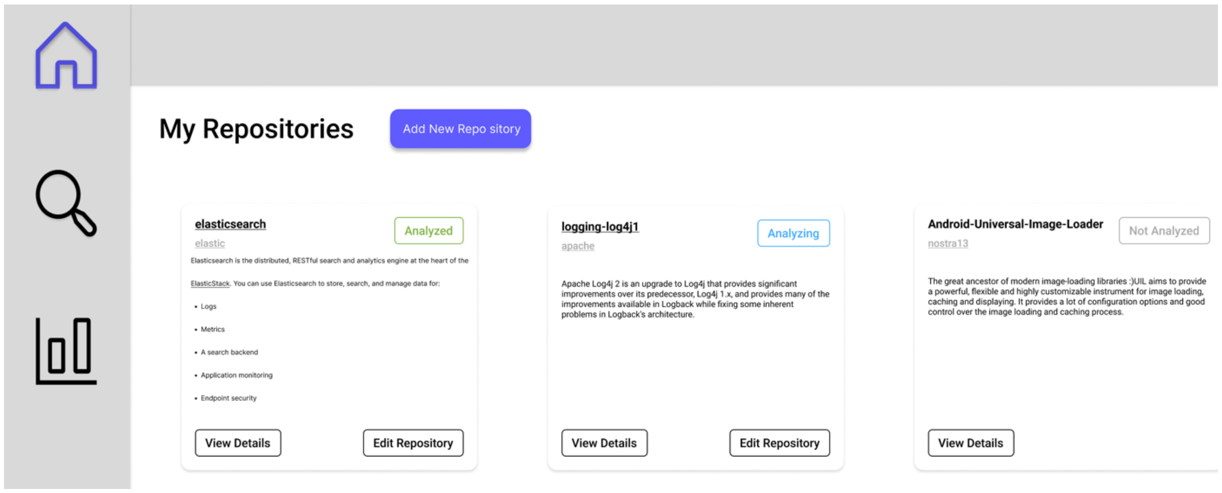
The central site keeps a list of alive sites in the federated network by recording their heartbeats. Each site is expected to send a heartbeat every hour. If a site's heartbeat is missing necessary notifications are performed. The central site also holds summarized metrics for the whole network, such as overall defect resolve velocity and its trend over some time.

C. User Interface Design

The user interface design is presented via Figures 3-5. A user either in a site or in the central site can see the repositories with proper access rights, as shown in Figure 3. To mimic this operation, Figure 1 presents some public GitHub repositories. This project repository and selection window also indicates the status of the project with four states: “Not Analyzed”, “Analyzing”, “Analyzed”, and “Failed”. After selecting a project, a window like the one in Figure 4 is shown and if the status is neither “Analyzing” nor “Analyzed”, the "Analyze" button appears. If it has already been analyzed, the details of the analyze operation are shown. To see the metrics, the metrics button should be pressed, and it takes the user to a window like the one shown in Figure 5. It is called the dashboard and presents various metrics with charts and graphs. Metrics, charts, and graphs are all customizable.

IV. RELATED WORK

The concept of federated networks is not new, and they are not limited to a certain field. The services are called federated if their service architecture spans numerous independent control domains [15]. It is challenging to manage federated services and provide effective customer assistance since only a tiny portion of the environment can be monitored and controlled by any given authority. Bhoj et al. [15] characterized many facets of federated networks as early as 1997. Some other examples of federated networks are as follows. For instance, Afsarmanesh et al. [16] proposed the PRODNET architecture for federated information management. Another example is Open Cirrus [17], which is proposed to federate a multitude of sites with diverse hardware, services, and tools for providing federated data centers for open source systems and services research. The sites reside on different continents and are subject to different privacy legislation and concerns.



elasticsearch — elastic — Analyzed

Elasticsearch is the distributed, RESTful search and analytics engine at the heart of the ElasticStack. You can use Elasticsearch to store, search, and manage data for:

- Logs
- Metrics
- A search backend
- Application monitoring
- Endpoint security

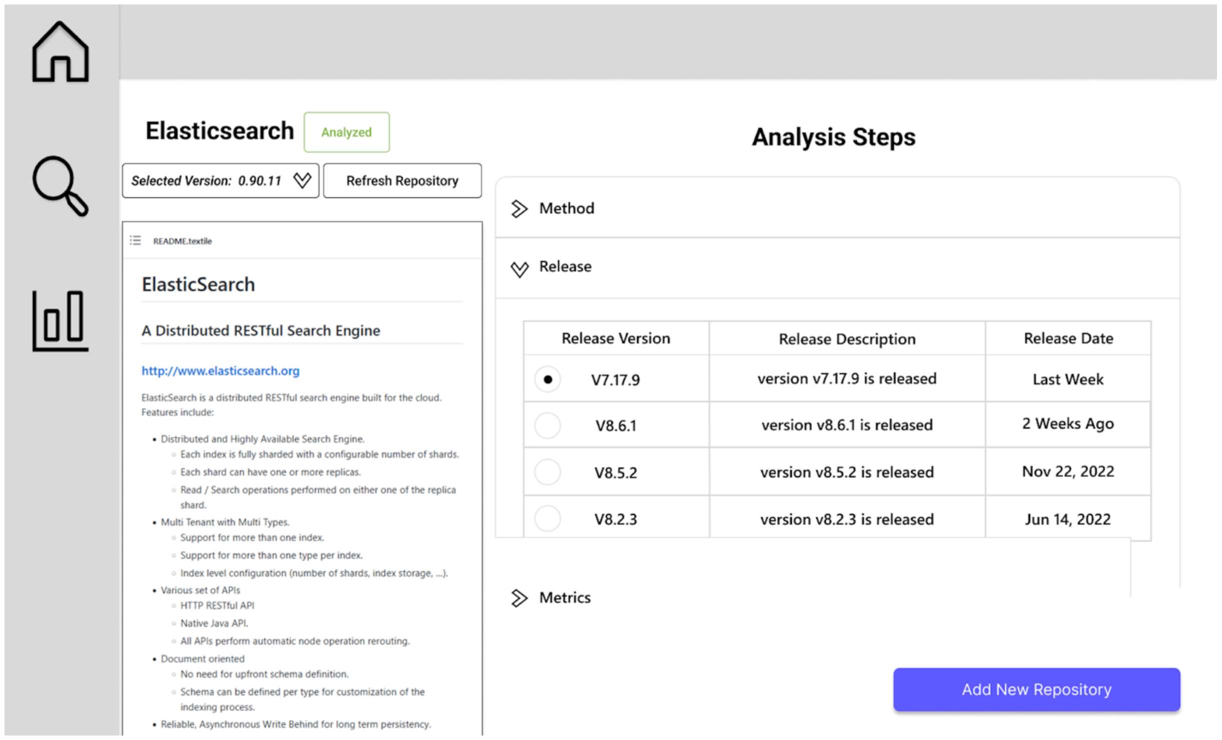
logging-log4j1 — apache — Analyzing

Apache Log4j 2 is an upgrade to Log4j that provides significant improvements over its predecessor, Log4j 1.x, and provides many of the improvements available in Logback while fixing some inherent problems in Logback's architecture.

Android-Universal-Image-Loader — nostra13 — Not Analyzed

The great ancestor of modern image-loading libraries :)UIL aims to provide a powerful, flexible and highly customizable instrument for image loading, caching and displaying. It provides a lot of configuration options and good control over the image loading and caching process.

Figure 3. Project repository and selection user interface design.



ElasticSearch

A Distributed RESTful Search Engine

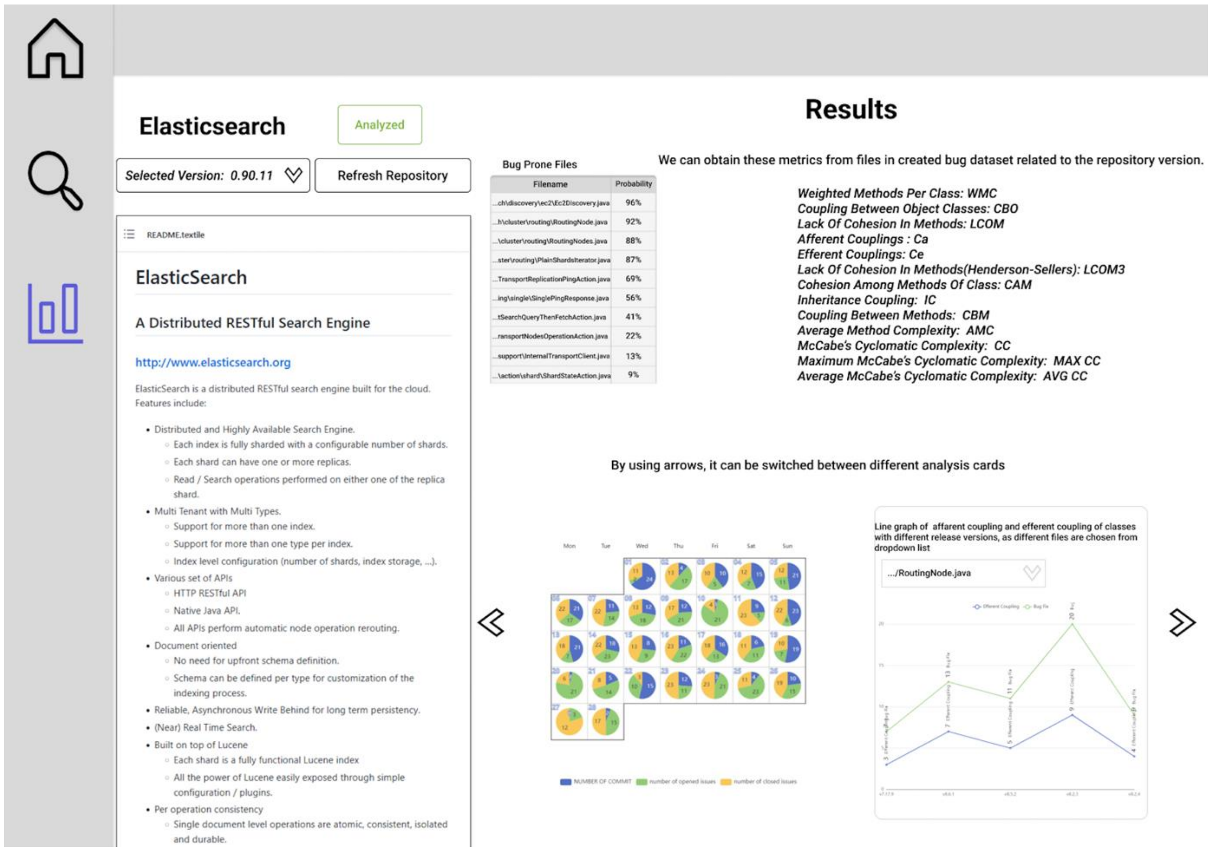
http://www.elasticsearch.org

ElasticSearch is a distributed RESTful search engine built for the cloud. Features include:

- Distributed and Highly Available Search Engine.
  - Each index is fully sharded with a configurable number of shards.
  - Each shard can have one or more replicas.
  - Read / Search operations performed on either one of the replica shard.
- Multi Tenant with Multi Types.
  - Support for more than one index.
  - Support for more than one type per index.
  - Index level configuration (number of shards, index storage, ...).
- Various set of APIs
  - HTTP RESTful API
  - Native Java API.
  - All APIs perform automatic node operation rerouting.
- Document oriented
  - No need for upfront schema definition.
  - Schema can be defined per type for customization of the indexing process.
- Reliable, Asynchronous Write Behind for long term persistency.

	Release Version	Release Description	Release Date
●	V7.17.9	version v7.17.9 is released	Last Week
○	V8.6.1	version v8.6.1 is released	2 Weeks Ago
○	V8.5.2	version v8.5.2 is released	Nov 22, 2022
○	V8.2.3	version v8.2.3 is released	Jun 14, 2022

Figure 4. Project analysis user interface design.



Elasticsearch

Selected Version: 0.90.11

Refresh Repository

README.textile

ElasticSearch

A Distributed RESTful Search Engine

http://www.elasticsearch.org

ElasticSearch is a distributed RESTful search engine built for the cloud. Features include:

- Distributed and Highly Available Search Engine.
  - Each index is fully sharded with a configurable number of shards.
  - Each shard can have one or more replicas.
  - Read / Search operations performed on either one of the replica shard.
- Multi Tenant with Multi Types.
  - Support for more than one index.
  - Support for more than one type per index.
  - Index level configuration (number of shards, index storage, ...).
- Various set of APIs
  - HTTP RESTful API
  - Native Java API.
  - All APIs perform automatic node operation rerouting.
- Document oriented
  - No need for upfront schema definition.
  - Schema can be defined per type for customization of the indexing process.
- Reliable, Asynchronous Write Behind for long term persistency.
- (Near) Real Time Search.
- Built on top of Lucene
  - Each shard is a fully functional Lucene index
  - All the power of Lucene easily exposed through simple configuration / plugins.
- Per operation consistency
  - Single document level operations are atomic, consistent, isolated and durable.

Results

Bug Prone Files

Filename	Probability
...ch\discovery\ec2\Ec2Discovery.java	96%
...h\cluster\routing\RoutingNode.java	92%
...\cluster\routing\RoutingNodes.java	88%
...ster\routing\PlainShardsIterator.java	87%
...TransportReplicationPingAction.java	69%
...ing\single\SinglePingResponse.java	56%
...tSearchQueryThenFetchAction.java	41%
...ransportNodesOperationAction.java	22%
...support\InternalTransportClient.java	13%
...\action\shard\ShardStateAction.java	9%

We can obtain these metrics from files in created bug dataset related to the repository version.

Weighted Methods Per Class: WMC
Coupling Between Object Classes: CBO
Lack Of Cohesion In Methods: LCOM
Afferent Couplings : Ca
Efferent Couplings: Ce
Lack Of Cohesion In Methods(Henderson-Sellers): LCOM3
Cohesion Among Methods Of Class: CAM
Inheritance Coupling: IC
Coupling Between Methods: CBM
Average Method Complexity: AMC
McCabe's Cyclomatic Complexity: CC
Maximum McCabe's Cyclomatic Complexity: MAX CC
Average McCabe's Cyclomatic Complexity: AVG CC

By using arrows, it can be switched between different analysis cards

Line graph of affarent coupling and efferent coupling of classes with different release versions, as different files are chosen from dropdown list

.../RoutingNode.java

Figure 5. Dashboard user interface design.

The health domain is currently running federated networks. For instance, CanDIG [18] is a Canadian national health federated research data platform designed to assist the finding, querying, and analysis of permitted health research data across institutions and projects. CanDIG is the first Canadian federation of many human genomes and biomedical data projects. Another proposal for health domain is the Cross-Institutional Clinical Translational Research project [19], which investigated a federated query tool and examined how this tool can facilitate the discovery of clinical trial cohorts by controlling access to aggregate patient data housed in academic medical centers that are not linked.

V. CONCLUSION

Each day, new features are added to software, and with each new feature, extra bugs may be introduced, and source quality may suffer. The scenario becomes more complicated if the software development is distributed with specific privacy and trade secret considerations. When addressing the challenges mentioned above, it is desired that the software quality be maintained above a particular threshold. Toward this goal, this paper proposes a federated source code quality query and analysis platform called FSCQQA.

With the proposed platform, sites are not required to disclose their codes with any other site while aiming for high source code quality and low defect ratio. At each site, local defect datasets will be generated and analyzed. The analysis results as defect metrics and the source code metrics obtained from the static analysis will be shared within the federated network and can be queried. Furthermore, trend analysis can be conducted at the central site and shared with consortium sites.

As future work, federated analytics and prediction using the local datasets are planned. The sites may push defect-related features to the central site for future machine learning. Such a defect database is valuable in terms of following the reliability of each site but also in improving defect-free development by providing in-depth analysis, such as root-cause analysis, and suggesting training and education. Then, the prediction model will generate predictions on sites. The prediction model will be updated and enhanced based on further coming data, meaning new source code. As developer data will not be exchanged, there will be no privacy concerns.

ACKNOWLEDGMENT

The authors would like to thank Univera Inc. for valuable guidance.

References

[1] "Networked European Software and Services Initiative-support office team." https://cordis.europa.eu/project/id/034359 (accessed Mar. 19, 2023).

[2] "Methodology and supporting toolset advancing embedded systems quality." https://cordis.europa.eu/project/id/286583 (accessed Mar. 19, 2023).

[3] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for java," presented at the Proceedings of the 14th international conference on predictive models and data analytics in software engineering, 2018, pp. 12–21.

[4] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for java and its assessment regarding metrics and bug prediction," *Software Quality Journal*, vol. 28, pp. 1447–1506, 2020.

[5] "Unified Bug Dataset." http://www.inf.u-szeged.hu/~ferenc/papers/UnifiedBugDataSet/ (accessed Mar. 19, 2023).

[6] H. Osman, M. Ghafari, O. Nierstrasz, and M. Lungu, "An extensive analysis of efficient bug prediction configurations," presented at the Proceedings of the 13th international conference on predictive models and data analytics in software engineering, 2017, pp. 107–116.

[7] E. Mashhadi, S. Chowdhury, S. Modaberi, H. Hemmati, and G. Uddin, "An Empirical Study on Bug Severity Estimation Using Source Code Metrics and Static Analysis," *arXiv preprint arXiv:2206.12927*, 2022.

[8] S. M. Henry and C. L. Selig, *Design Metrics which Predict Source Code Quality*. Department of Computer Science, Virginia Polytechnic Institute and State University, 1987.

[9] T. Pearse and P. Oman, "Maintainability measurements on industrial source code maintenance activities," presented at the Proceedings of International Conference on Software Maintenance, IEEE, 1995, pp. 295–303.

[10] K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and application of an automated source code maintainability index," *Journal of Software Maintenance: Research and Practice*, vol. 9, no. 3, pp. 127–159, 1997.

[11] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, "Source code metrics: A systematic mapping study," *Journal of Systems and Software*, vol. 128, pp. 164–197, 2017.

[12] "GitHub." https://github.com/ (accessed Mar. 19, 2023).

[13] Department of Software Engineering, University of Szeged, Hungary, "OpenStaticAnalyzer." https://openstaticanalyzer.github.io/ (accessed Mar. 19, 2023).

[14] "OPENAPI Initiative." https://www.openapis.org/ (accessed Mar. 19, 2023).

[15] P. Bhoj, D. Caswell, S. Chutani, G. Gopal, and M. Kosarchyn, "Management of new federated services," presented at the Integrated Network Management V: Integrated management in a virtual world Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management San Diego, California, USA, May 12–16, 1997, Springer, 1997, pp. 327–340.

[16] H. Afsarmanesh, C. Garita, Y. Ugur, A. Frenkel, and L. O. Hertzberger, "Design of the federated information management architecture for PRODNET," presented at the Infrastructures for Virtual Enterprises: Networking Industrial Enterprises IFIP TC5 WG5. 3/PRODNET Working Conference on Infrastructures for Virtual Enterprises (PRO-VE'99) October 27–28, 1999, Porto, Portugal 1, Springer, 1999, pp. 127–146.

[17] R. H. Campbell *et al.*, "Open Cirrus^{TM} Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research.," *HotCloud*, vol. 9, pp. 1–1, 2009.

[18] L. J. Dursi *et al.*, "CanDIG: Federated network across Canada for multi-omic and health data discovery and analysis," *Cell Genomics*, vol. 1, no. 2, p. 100033, 2021.

[19] N. Anderson *et al.*, "Implementation of a deidentified federated data network for population-based cohort discovery," *Journal of the American Medical Informatics Association*, vol. 19, no. e1, pp. e60–e67, 2012.