Predicting Future Change Requests in Agile Software Engineering

Samir Omanovic, Emir Buza

Abstract**—**Software engineering based on agile methods is different than plan driven in many aspects. Based on our practical experience in agile software engineering we concluded that one of the most important success factors is predicting future change requests. This article emphasizes importance of the future change requests frequency as a very important analysis factor for the later solution selection and software maintenance. It describes a positive experience related to the agile software engineering of the software system for the data import in an environment with frequent change requests, through a case study. The main reason for the success is that the estimation about future changes is taken into account during the analysis. Data import is based on the web service for the XML upload and Oracle database objects for importing, storing and checking data. Meta-model based design is applied to gain flexibility and meet customer's frequent change requests. A change request is implemented through changing the meta-model parameters which is fast and reliable. There were many change requests through the life of this software system and all of them where low cost changes. Initial higher cost to develop the software that is easy changeable is reimbursed later during the software evolution. Also, changes are implemented fast, with a minimum effort, with the high quality and with the high customer satisfaction.

Keywords**—**Agile software engineering, database, meta-model, software change management.

I. Introduction

AGILE software engineering is different than plan driven in many aspects. Fig. 1 shows a simple comparison of agile and plan driven software engineering [1]. If we are building a bridge from point A to point B then it is normal that we know exactly the starting and ending point of the bridge – plan driven engineering. If we imagine that we are building the same bridge using agile methods then our ending point is not exactly known and we correct our estimation about the ending point in each iteration. For that reason, final bridge that is built using agile method is not straight – it is waved.

Based on our practical experience in agile software engineering we concluded that one of the most important success factors is predicting future change requests – making good prediction, not only of point B, but also of point C (see fig.1.). Point C represents a state of the software after some period of the maintenance.

There are many interesting works related to agile methods that are focused on specific methods (for example [2][3][4][5]). This article gives general conclusions related to all agile methods.

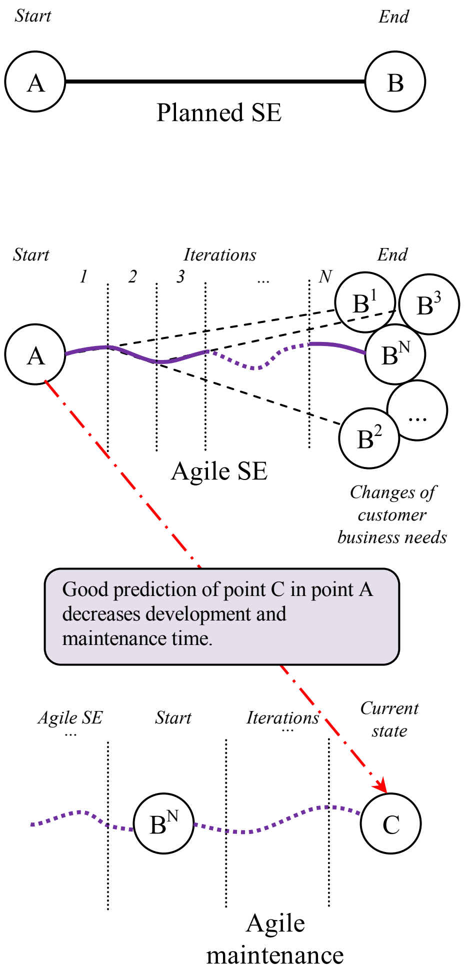


Fig. 1 Agile vs plan driven software engineering [1]

Agile maintenance [6] and dynamic business environment often requires highly flexible solutions. It is very important to

Samir Omanovic is with the Faculty of Electrical Engineering in Sarajevo - Department for Computer Science and Informatics, Zmaja od Bosne bb, Kampus Univerziteta, 71000 Sarajevo, Bosnia and Herzegovina (e-mail: samir.omanovic@etf.unsa.ba).
Emir Buza is with the Faculty of Electrical Engineering in Sarajevo - Department for Computer Science and Informatics, Zmaja od Bosne bb, Kampus Univerziteta, 71000 Sarajevo, Bosnia and Herzegovina (e-mail: emir.buza@etf.unsa.ba).

estimate how often change requests will be issued in the future – during the software maintenance [7], and have that in mind when choosing a solution from the domain of solutions [8]. Most important is to have in mind systems theory [9] and observe the software system in its environment (see fig.2). Forces from the environment (business environment in this case) represented as change requests modify software system. If these forces are well predicted then software system design will include many parameters in points of change and enable adaptation of the software system with minimum effort. Software system must have some level of flexibility to support agile maintenance and satisfy business needs in efficient way.

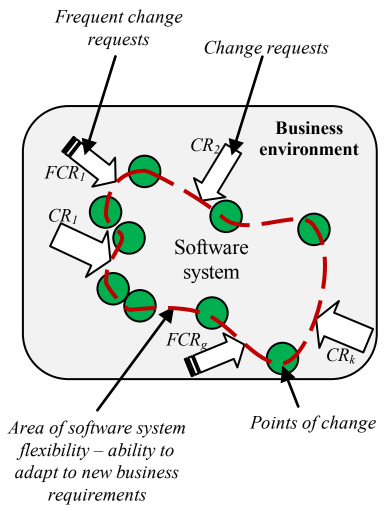


Fig. 2 Software system in its environment [1]

This article emphasizes importance of the future change requests frequency factor [10] as a very important analysis factor for the later solution selection and software maintenance. It describes a positive experience related to the software engineering of the software system for the data import in an environment with frequent change requests, through a case study. The main reason for the success is that the estimation about future changes is taken into account during the analysis phase [8][11].

II. Problem Formulation

In this project, the customer expressed a need to have a data import system which will integrate the point of sale (POS System) system and the central production system (Core System). It was requested that the data are transferred using XML [12] files, defined by the XSD schema [12] and Java based web service [13] for upload of files.

A. Problem domain

At the start of each project we are faced with the project definition phase [8] and as the first step we have to identify the problem domain [8].

Requested process is described as: The POS System calls the web service and passes the XML file with new data that should be placed in the Core System. Data Import System temporary stores data and validates them according to the XSD schema. Data that passed validation are placed in the Core System. The rest of data stays in a temporary storage until they are corrected to become valid, or deleted by the user that controls the Data Import System.

B. Problem analysis

Without having a wider perspective and based on the given process description, this looks like a pretty simple problem for which it is easy to make a solution (for example – create a Java based web service [13] that will perform import, validation and storing data in the database [14]).

But, having the experience in the maintenance of the Core System for this customer, it was estimated that future change requests frequency can become a problem. Core System is constantly growing. POS System must follow that. Finally, the Data Import System must be enough flexible to adopt and follow the growth of the Core System and the POS System. Software evolution [11] is very intensive.

This is all related to costs [11] of implementing change requests in the Data Import System. It is not logical, nor acceptable to the customer that cost of a change on an auxiliary system – the Data Import System, is higher than the cost of the correspondent changes on the Core System and the POS System.

C. Solution selection

Based on the quick analysis of the problem, it was concluded that the change in data transfer like adding a new column or adding a new XSD schema must be as simple as changing some easily accessible parameters with minimum interventions on the source code. If there is a need to change the code then is better that the change is done on some simple stored procedure than on the web service application.

Solutions like having most of the processing on the web service or relaying mostly on the database XML support [15] are rejected as inadequate regarding the estimated change requests frequency.

It is decided that most of the processing should be on the database, and that tables and stored procedures should be used to create the meta-model [16] (see also metadata [17]) that describes the model of the process, necessary transformations and validations and which will give the adequate flexibility related to future high change requests frequency. This is very similar concept to the data staging area in the data warehousing [17], which is “a storage area and a set of processes commonly referred to as extract-transformation-load (ETL)” [17]. In this case we can name this as extract-validate-load.

To gain the stated goal, the model of the process should be highly parameterized and those parameters should be changeable easily.

D. Solution constraints

Constraints related to the solution where caused by existing systems. Core System worked on a database Oracle 8.1.7. [18] and the Data Import System also had to work on the same database. POS System was based on Java and used Apache Tomcat [19], so it is decided that the Data Import System's web service must also be based on Java and work on Apache Tomcat. Additionally, for better performances in working with large files, the web service should use Message Transmission Optimization Mechanism (MTOM) [20] and Apache Axis2/Java [21].

III. System Description

Fig.3. shows the overall picture of the main idea. The main concern was how to achieve low cost and fast changes. Aspect of change management [11] was very important.

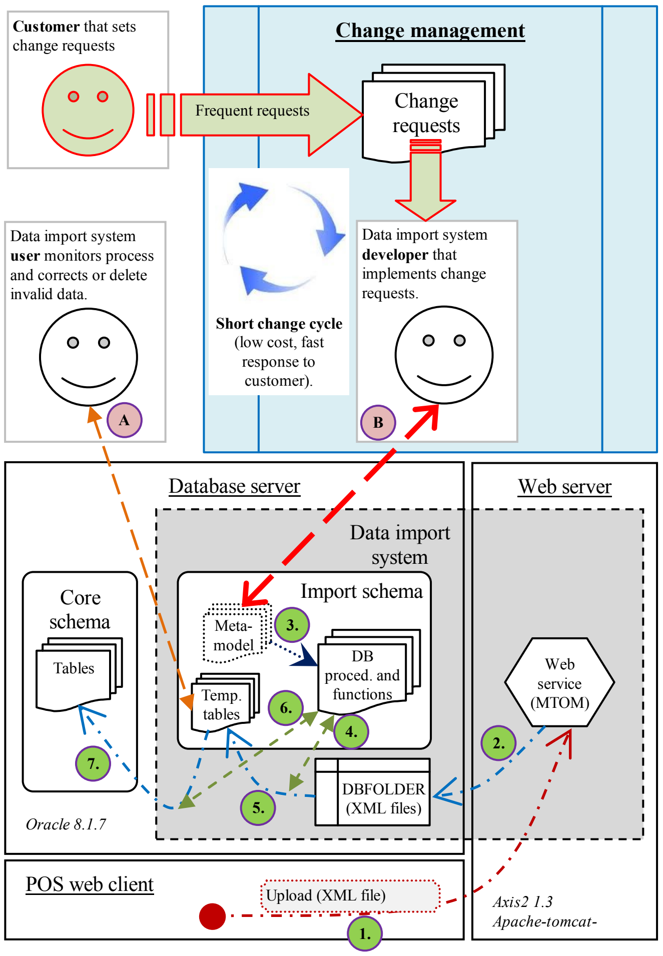


Fig. 3 Data Import System (initial idea) [10]

Fig.4 shows the main process – Data Import System activity diagram [22]. At the start, the POS System calls the web service and passes two parameters (step 1). First parameter is the XML file and the second parameter is the XSD schema name. Web service accepts the XML file and stores it in the shared database folder (step 2). After that, the web service calls the database procedure for import and passes two parameters – the XML file name and the XSD schema name. Based on the XSD schema name, meta-model parameters are taken and the import procedure is configured (step 3). Import procedure validates data against XSD schema rules (stored as parameters of the meta-model) and then dynamically creates INSERT statements based on mappings between XML tags and columns/tables in the Import schema (also stored as parameters of the meta-model; step 4). Created statements are executed and tracing information is added into log tables (step 5). It is possible that some statement fails during execution, because it breaks some constraint on temporary tables. Information about these errors is also placed into log tables. After importing all data from the XML file, the database procedure that validates data against business rules is called and all valid data are transferred in the Core System. Invalid data are left in temporary tables and they are handled by the user that monitors the process and corrects or deletes invalid data (process A on the fig.3).

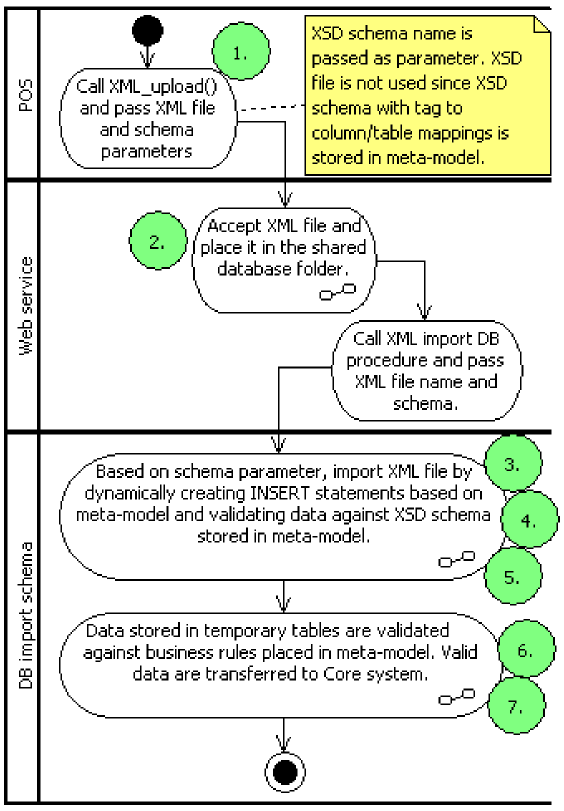

POS
Web service
DB import schema
Call XML_upload() and pass XML file and schema parameters
XSD schema name is passed as parameter. XSD file is not used since XSD schema with tag to column/table mappings is stored in meta-model.
Accept XML file and place it in the shared database folder.
Call XML import DB procedure and pass XML file name and schema.
Based on schema parameter, import XML file by dynamically creating INSERT statements based on meta-model and validating data against XSD schema stored in meta-model.
Data stored in temporary tables are validated against business rules placed in meta-model. Valid data are transferred to Core system.
1.
2.
3.
4.
5.
6.
7.

Fig. 4 Data Import System activity diagram [10]

Most of the rules related to columns format and sizes are in the XSD schema. In this system they are placed in the meta-model tables. Besides that, the meta-model contains mappings of XML tags and columns/tables in the import schema. Those mappings enable dynamic INSERT statement creation.

Procedure that moves data from temporary tables to the Core System checks business rules. For example, the Core System has strict evidence of issued documents and if the document arrived from the POS is not registered in the strict evidence it will not pass to the Core System. User can place that document number in the strict evidence on the Core System and after that the document will move to the Core System. Document in this context represents the data from several tables. For example, an invoice has a header in one table and items in the other table.

Beside business rules, this procedure detects duplicate documents (documents that are already imported in the Core System) based on the content received in the XML file. This functionality is important when POS sends same document multiple times. Documents from the received XML file are validated (validation of data types, data sizes, etc.) and placed in temporary tables. Procedure then checks is there records in temporary tables which constitute one document that is already transferred to the Core System (based on primary keys). Duplicates are automatically deleted from temporary tables and information is recorded in log tables.

POS System can send incomplete documents. This procedure implements higher abstraction validation and if document has no all necessary parts, contained in several tables, then it is not moved to the Core System.

User has an interface for viewing the invalid data (data that didn’t pass different validations). That interface provides possibility to change some of invalid data and to start the database procedure for moving data from temporary tables to the Core System. With that possibility, the user can make the most correction on the problematic data. Also, there is a possibility to view log tables content and identify cause of problems (import problems or move problems). Dynamically created import statements whose execution failed are stored in log for the future analysis.

IV. Meta-Model Description

Meta-model designed for this system describes data formats and business rules for the process of importing XML files. Any change in the meta-model automatically reflects on the process of importing XML files. This represents a set of highly parameterized points of change in software which is in line with the high flexibility of the software system. That enables change implementation with the minimum effort, minimum risk and high customer satisfaction.

The heart of the meta-model includes tables named `XML_COLUMNS` and `XML_TABLES`.

Information stored in `XML_COLUMNS` table is used for dynamic creation of the INSERT statement using database function named `CREATE_INSERT_STATEMENT`. Columns `TABLENAME` and `COLUMNNAME` define in which temporary table are stored data and column `XMLTAGNAME` defines from which XML attribute to read data. This is the mapping of columns and XML attributes. Adding, modifying or deleting records in the `XML_COLUMNS` table automatically changes the process of importing XML files.

Information stored in `XML_TABLES` table enables mapping of temporary tables and XML tags. Also it represents the structure of one document – invoice for example, which should be checked (missing parts, wrong code from the codebook, etc.)

This enables two level of validation. The first level is during the reading of the XML file when it is possible to use information from the table `XML_COLUMNS` to check data types, data sizes, etc. This is analogous to the XSD schema validation. The second level of validation is during inserting the statement created by the function `CREATE_INSERT_STATEMENT`, where the insert can fail if do not pass the database validation on inserting into temporary tables. These validations are related to data relations and document context. All insert statements that do not pass the second level of the validation are stored in the `IMPORT_ERRORS` table with the complete insert statement and error description for the later analysis.

Table `XML_TABLES` has a column `FILENAME` which is used for defining different import processes. For example, one process is importing XML files with invoices and the other process is importing XML files with contracts. Each client application in these processes (in his call to the web services) passes his type as parameter – `InvoiceClient` or `ContractsClient` for example. This column is also used as a parameter for differencing imports of the same documents in different contexts. For example, invoices for persons (`InvoicePerson`) and invoices for companies (`InvoiceCompany`) are passing different validation, although they are placed in the same target tables in the Core System.

Stored procedures like `IMPORT_XML`, `MOVE_DATA`, `DELETE_DOC_FROM_TMP` and others are using information stored in `XML_COLUMNS`, `XML_TABLES` and other tables to dynamically modify their behavior according to current settings.

V. Software System Evolution

System evolution is tracked through time and there were many changes on the Data Import System and they can be categorized as:

A. Adding a new table(s) or a column(s) in the existing XSD schema.

B. Changing column(s) properties in the existing XSD schema.

C. Changing business rules related to moving data from temporary tables to the Core System.

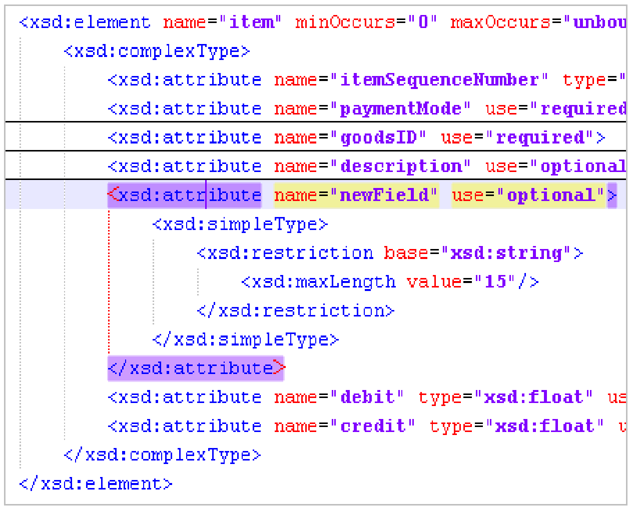
D. Adding a new XSD schema in transfer.

E. Other change requests.

A. Adding new table(s) or column(s) in the existing XSD schema

This type of change is most frequent. Meta-model enables easy implementation since adding new mappings in the meta-model will enable that the import procedure can import the XML file with additional column(s) and/or table(s). Also it is necessary to alter existing temporary tables with new columns and/or add new temporary tables that are mapped. This is small change that requires small effort. It is very fast and costs low. That brings customer's high satisfaction.

POS System uses the XSD schema for the validation of XML files before upload. Data Import System do not use the XSD schema file since elements contained in the XSD schema file are placed in the database (in the meta-model parameters). When a new column is added in the POS System then XSD file and the POS application are changed. Same column in the Core System is added by altering the table and changing the Core applications. In the Data Import System this change is implemented by adding a single row in the XML_COLUMNS table and by altering a single temporary table. No change on the upload service or any other part of the system is required. In the example on fig.5, a new column named **`newField`** is added. Let us assume that the corresponding table column (we added it by altering the table) is named **`NEW_FIELD`** in the table **`TMP_TABLE1`**.



```
<xsd:element name="item" minOccurs="0" maxOccurs="unbou
    <xsd:complexType>
        <xsd:attribute name="itemSequenceNumber" type="
        <xsd:attribute name="paymentMode" use="required
        <xsd:attribute name="goodsID" use="required">
        <xsd:attribute name="description" use="optional
        <xsd:attribute name="newField" use="optional">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="15"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="debit" type="xsd:float" us
        <xsd:attribute name="credit" type="xsd:float" u
    </xsd:complexType>
</xsd:element>
```

Fig. 5 Example part of the XSD schema after the change of type A [10]

Then we add a new record to the **`XML_COLUMNS`** table using the following statement (or editing the table using some tool which is more convenient):

```
INSERT INTO XML_COLUMNS (
  TABLENAME, COLUMNNAME
  , XMLTAGNAME, XMLTAGTYPE
  , … )
VALUES (
  'TMP_TABLE1', 'NEW_FIELD'
  , 'newField', 'string'
  , … )
```

This change will cause that the function `CREATE_INSERT_STATEMENT` will automatically include this new column in the insert statement creation. Procedures for importing XML and moving data from temporary tables to the Core System will also use this new column.

Change described in this example can be done in a few minutes and does not require change in the web service source code or restart of the application or any other complicated intervention.

B. Changing column(s) properties in the existing XSD schema

These are less frequent change requests than the type A. They can be easily implemented in two ways:

1) by changing XSD schema stored in the meta-model, or
2) by altering temporary tables and changing column properties.

In both ways, data that do not pass XSD schema validation will be stopped before inserting into temporary tables. Change requires small effort. It is very fast and costs low.

C. Changing business rules

These are approximately same frequent change requests like the type B. They are implemented by altering column properties in temporary tables or by altering stored procedure for moving data from temporary tables to the Core System. Change requires small effort. It is very fast and costs low.

D. Adding new XSD schema in transfer

This type of change requests is rare. Meta-model enables easy implementation – adding new mappings in the meta-model will enable that import procedure can import XML file with additional column(s) and/or table(s). Also it is necessary to create temporary tables for a new schema, change database procedure that validates and moves data from temporary tables to the Core System. Change requires moderate effort, but still is fast and costs low.

E. Other change requests

More than 90% of change requests were of types A, B, C or D. During more than 3 years of the life of the system and 107 change requests, there were only one change request that involved change on the web service. This change request was to enable data import in 3 different databases (instances). This change added one more parameter in the call to the web service – database ID.

This change request can be treated like a normal change request that is not for the fast implementation.

Sometimes, the customer requests that some small automation related to data corrections is implemented. There were only few change requests of this type. For example, it is necessary to change the value of some data under the predefined condition. These changes require small effort. They are very fast and they cost low. One of the ways to implement such a request is to add/modify trigger on the corresponding temporary tables.

Few times, customer needed export of some data from the Core System in the XML file, based on the XSD schema used for import. For that purpose, database stored procedure for exporting from the Core System to the XML is created as a reverse process of importing and is using the same meta-model mappings of columns/tables and XML tags (see appendix 1 - example function `get_col_select`). If `get_col_select` function is executed during export process with some parameters – for example:

```
SELECT getcolselect (
         'VALUTA'
       , 'currencyCodes'
       , 0
       , 'Codebooks'
       , 1)
  FROM DUAL;
```

Then it will dynamically form SELECT statement like:

```
SELECT '<currencyCodes '
       || REPLACE (
           REPLACE (
            REPLACE (
             REPLACE (
              'currencyCodeN="'
              || REPLACE (
                   valuta
                 , '"'
                 , '&' || 'quot;'
                 )
              || '"  name="'
              || REPLACE (
                   naziv
                 , '"'
                 , '&' || 'quot;'
                 )
              || '"  currencyCodeA="'
              || REPLACE (
                   kratica
                 , '"'
                 , '&' || 'quot;'
                 )
              || '"  measure="'
              || REPLACE (
                   mjera
                 , '"'
                 , '&' || 'quot;'
                 )
              || '"  ', '&', '&' || 'amp;'
             ), '''', '&' || 'apos;'
            ), '<', '&' || 'lt;'
           ), '>', '&' || 'gt;'
          ) || '/>'
  FROM valuta
```

Execution of the created statement will generate part of the XML structure that is incorporated in the final XML file. In this example it will generate part of XML like:

```
<currencyCodes measure="1"
   currencyCodeA="AUD"
   name="AUSTRALIJSKI DOLAR"
   currencyCodeN="36"/>
<currencyCodes measure="100"
   currencyCodeA="ATS"
   name="ŠILING"
   currencyCodeN="40"/>
...
```

Any change in the table will automatically reflect behavior of this function. This functionality is now in use more than it was initially planned, for exporting codebooks and documents from the Core System for use by the POS System.

VI. Advantages of the Selected Solution

Selected solution showed many advantages in the agile maintenance process during more than 3 years of life of the system. Customer is highly satisfied with the provided solution, in all aspects, especially with the fast response to change requests.

Comparison of effort needed for the change requests on the Core System or the POS System on one side and effort needed for the correspondent change requests on the Data Import System, on the other side, shows us that the Data Import System software evolution requires much less effort.

Generally the most important advantage of the Data Import System is simple software evolution, but there are some other advantages like:

1) It is possible to import XML files without using the web service (off-line import). At start, this was an undocumented feature [23], but later it is used to import XML files in the situations when the POS System failed to communicate to the web service. It is also used for testing purposes.
2) It is not dependant on Oracle database version. There was a database upgrade from Oracle 8.1.7 to Oracle 10g and it was not necessary to change the system, since most of database objects are custom-made.
3) User that monitors the Data Import System has the possibility to correct most of the problems with import. If he spots some pattern in data errors he can request introducing some automation related to data correction.
4) High expandability – it is now in use for data exchange with additional external systems like Web Shop e-commerce and Bank cross-selling [24], where data exchange is performed in the same way like with the POS System.

VII. Conclusion

Usually, when selecting the solution for the given problem, **the future change requests frequency factor** is not treated as an important factor. Actually, in most analyses is not taken into account at all. This success story shows that this factor is very important, especially in the agile software development.

Taking this factor into account will probably lead to the solution that initially costs more (using more process meta-modeling, selecting solution for the class of problems instead for the one problem instance, engagement of experienced software engineers, etc.), but long-term costs will be evidently decreased. **The change implementation time is sometimes more important for the business than the change implementation costs.**

All described advantages are related to the customer benefit, but there are also advantages for the software engineering company. **Simplified software maintenance requires less skilled developers. There is less points of change in software which reduces possibility to have a bug in a release. Software testing is simplified.**

Flexible solution supports both – the agile maintenance and fast business response. Agile concept is a result of a necessity to have fast business response – to be agile in business. That means that software solutions for business should have high flexibility, especially in the context when maintenance is based on agile methods.

Appendix

A. Appendix 1 - example function get_col_select

```
CREATE OR REPLACE FUNCTION get_col_select (
   v_tablename      IN    VARCHAR2,
   v_tagname        IN    VARCHAR2,
   v_tab_num        IN    INT,
   v_filename       IN    VARCHAR2,
   v_close          IN    INT
   -- 1 Close tag, 0 Leave tag opened
)
   RETURN VARCHAR2
AS
   CURSOR column_cur
   IS
        SELECT columnname
             , xmltagname
             , xmltagtype
          FROM xml_columns
         WHERE UPPER (tablename)
             = UPPER (v_tablename)
           AND UPPER (file_name)
             = UPPER (v_filename)
           AND toexport = 'Y'
      ORDER BY orderby;

   v_select   column_cur%ROWTYPE;
   ret        VARCHAR2 (4000);
   ret1       VARCHAR2 (4000);
   tabs       VARCHAR2 (20);
   cnt        NUMBER;
BEGIN
   -- Uses the word XSELECT instead of SELECT
   -- to avoid leather wrong replacements.
   -- Before use of the statement XSELECT is
   -- replaced to SELECT.
   tabs :=
         CHR (9)
      || CHR (9)
      || CHR (9)
      || CHR (9)
      || CHR (9)
      || CHR (9)
      || CHR (9)
      || CHR (9)
      || CHR (9);
   ret1 :=
         'XSELECT '''
         || SUBSTR (tabs, 1, v_tab_num)
         || '<'
         || v_tagname
         || ' ';
   ret := '';

   OPEN column_cur;

   LOOP
      FETCH column_cur
       INTO v_select;

      EXIT WHEN column_cur%NOTFOUND;

     -- If column type is DATE
      IF (v_select.xmltagtype = 'D')
      THEN
         ret :=
               ret
            || v_select.xmltagname
            || '="'''
            || '||REPLACE(TO_CHAR('
            || v_select.columnname
            || ',''YYYY-MM-DD''),'
            || '''"'',''&''||'''
            || 'quot;'')||''"  ';
      -- If column type is DATETIME
      ELSIF (v_select.xmltagtype = 'T')
      THEN
         ret :=
               ret
            || v_select.xmltagname
            || '="'''
            || '||REPLACE(REPLACE(TO_CHAR('
            || v_select.columnname
            || ',''YYYY-MM-DD HH24:MI:SS''),'
            || ''' '',''T''),''"'',''&''||'''
            || 'quot;'')||''"  ';
      -- If column type is FLOAT
      ELSIF (v_select.xmltagtype = 'F')
      THEN
         ret :=
               ret
            || v_select.xmltagname
            || '="'''
            || '||REPLACE(numbertostring('
            || v_select.columnname
            || '),''"'',''&''||'''
            || 'quot;'')||''"  ';
      -- Column is treated as string
      ELSE
         ret :=
               ret
            || v_select.xmltagname
            || '="'''
            || '||REPLACE('
            || v_select.columnname
            || ',''"'',''&''||'''
            || 'quot;'')||''"  ';
      END IF;
   END LOOP;

   -- Replace special characters
   ret :=
         ret1
      || '''||REPLACE(REPLACE(REPLACE('
      || 'REPLACE('''
      || ret
      || ''',''&'',''&''||'''
      || 'amp;''),'''''''',''&''||'''
      || 'apos;''),''<'',''&''||'''
      || 'lt;''),''>'',''&''||'''
      || 'gt;'')||''  ';

   IF (v_close = 1)
   THEN
      ret :=
         SUBSTR (ret, 1, LENGTH (ret) - 2)
         || '/>'' FROM '
         || TRIM (v_tablename);
   ELSIF (v_close = 0)
   THEN
      ret :=
         SUBSTR (ret, 1, LENGTH (ret) - 2)
         || '>'' FROM '
         || TRIM (v_tablename);
   ELSE
      ret := '';
   END IF;

   RETURN SUBSTR (ret, 1, 4000);
END;
```

REFERENCES

[1] S.Omanovic, "Advanced Software Engineering" - course lecture notes, unpublished.

[2] E. Corona, F. E. Pani, "An Investigation of Approaches to Set Up a Kanban Board, and of Tools to Manage it" in *Proc. 11th WSEAS Conference on Telecommunications and Informatics (TELE-INFO '12)*, Saint Malo & Mont Sinat-Michel, 2012, pp. 53-58.

[3] V. Mahnic, "Introducing Scrum into the Development of a News Portal" in *Proc. 12th WSEAS International Conference on Applied Informatics and Communications (AIC '12)*, Istanbul, 2012, pp. 109-114.

[4] K. Fertalj, N. Hlupić, and D. Kalpić, "Permeation of RUP and XP on Small and Middle-Size Projects", in *Proc. 5th WSEAS International Conference on Telecommunications and Informatics (TELE-INFO '06)*, Istanbul, 2006, pp. 98-104.

[5] V. Mahnic, N. Zabkar, "Measurement repository for Scrum-based software development process", in *Proc. 2nd WSEAS Int. Conf on COMPUTER ENGINEERING and APPLICATIONS (CEA'08)*, Acapulco, 2008, pp. 23-28.

[6] M. Pronschinske. (2011, October 14). You can't be Agile in Maintenance? [Online]. Available: http://agile.dzone.com/news/you-can%E2%80%99t-be-agile-maintenance

[7] T. Hung VO. (2007, July 8). Software Maintenance [Online]. Available: http://cnx.org/content/m14719/1.1/

[8] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering* (2nd ed.). Pearson Prentice Hall, 2004.

[9] A.Habul, S.Omanovic, *Teorija sistema i informacija*, Sarajevo, Ekonomski fakultet, 2009.

[10] S.Omanovic, E.Buza, "Importance of Future Change Requests Frequency Analysis Factor for Solution Selection and Software Maintenance" in *Proc. 7th WSEAS European Computing Conference (ECC '13)*, Dubrovnik, 2013, pp. 246-251.

[11] I. Sommerville, *Software Engineering* (9th ed.). Addison-Wesley, 2010.

[12] w3schools.com. (2013). W3Schools Online Web Tutorials [Online]. Available: http://www.w3schools.com

[13] L. Sandakith. (2007, June 29). Eclipse WTP Tutorials - Creating Bottom Up Web Service via Apache Axis2 [Online]. Available: http://www.eclipse.org/webtools/community/tutorials/BottomUpAxis2WebService/bu_tutorial.html

[14] Rose India. (2012, September 20). Insert XML file data to database [Online]. Available: http://www.roseindia.net/tutorial/java/core/insertXMLFileDataToDatabase.html

[15] Oracle. (2000, Septembar). Oracle8i Application Developer's Guide – XML [Online]. Available: http://docs.oracle.com/cd/A87860_01/doc/appdev.817/a86030.pdf

[16] J. Mylopoulos (2004), Metamodeling [Online], Available: http://www.cs.toronto.edu/~jm/2507S/Notes04/Meta.pdf

[17] R.Kimball, M. Ross, *The Data Warehouse Toolkit - The Complete Guide to Dimensional Modeling* (2nd ed.). Wiley Computer Publishing, 2002.

[18] Oracle. Oracle8i 8.1.7 Documentation [Online]. Available: http://www.oracle.com/technetwork/documentation/oracle8i-085806.html

[19] Apache Software Foundation. Apache Tomcat [Online]. Available: http://tomcat.apache.org/

[20] The W3C website. SOAP Message Transmission Optimization Mechanism [Online]. Available: http://www.w3.org/TR/soap12-mtom/

[21] Apache Software Foundation. Apache Axis2/Java website [Online]. Available: http://axis.apache.org/axis2/java/core/

[22] D. Donko and S. Omanovic, *Objektno orijentirana analiza i dizajn primjenom UML notacije*. Sarajevo, Elektrotehnicki fakultet, 2009.

[23] Wikipedia. Undocumented feature [Online]. Available: http://en.wikipedia.org/wiki/Undocumented_feature

[24] Wikipedia. Cross-selling [Online]. Available: http://en.wikipedia.org/wiki/Cross-selling

S.Omanovic is born in Bosnia and Herzegovina, Visoko on the 1th of February in 1975. He received Dr.Sc. degree in 2011, Mr.Sc. degree in 2006, and graduate engineer degree in 2000, all in Computing and Informatics, from the Sarajevo University – Faculty of Electrical Engineering in Sarajevo, Bosnia and Herzegovina.

From 2000 to 2010 he has worked in several software engineering companies on different positions – software developer, team leader, senior software architect, and department leader. In parallel, from 2003 to 2010 he has worked part-time on the Faculty of Electrical Engineering in Sarajevo as assistant and senior assistant. From 2010 he works full-time on the Faculty of Electrical Engineering in Sarajevo and is now assistant professor and lecturer on master courses Advanced Software Engineering, and Pattern Recognition and Image Processing. Most of his faculty research is in the field of artificial intelligence, and most of his industry expertise is in the field of software engineering.

Dr. Omanovic's current research interests are in areas of pattern recognition, computer vision, image processing and advanced software engineering.

E.Buza is born in Bosnia and Herzegovina, Visoko on the 10th of September in 1977. He received Mr.Sc. degree in 2009, and graduate engineer degree in 2002, all in Computing and Informatics, from the Sarajevo University – Faculty of Electrical Engineering in Sarajevo, Bosnia and Herzegovina.

From 2003 to 2007 he has worked in software engineering company as senior software architect. In parallel, from 2003 to 2007 he has worked part-time on the Faculty of Electrical Engineering in Sarajevo as assistant. From 2007 he works full-time on the Faculty of Electrical Engineering in Sarajevo and is now senior assistant and assistant lecturer on courses Fundamental of Database Systems and Advance Database Systems. Most of his faculty research is in the field of data mining and bioinformatics, and most of his industry expertise is in the field of software engineering.

Mr Sc. Buza's current work is mostly on his doctoral thesis that includes research related to bioinformatics, data mining, artificial intelligence and software engineering.