Selective Validation of Test Results in a DotNet Project

by

Harry M. Sneed (MPA)
Software Test Engineer
ANECON GmbH, Vienna, Austria
Harry.Sneed@T-online.de

Abstract: The following paper describes a tool supported approach to the functional testing of a DotNet system at the Austrian Chamber of Commerce. The new system for collecting membership dues was intended to replace an existing mainframe system which had been in operation for over 20 years. The approach taken was that of selective validation. The database contents of the new SQL databases were selectively compared to the contents of the old VSAM files while the contents of the new XML output documents were compared against the contents of the old printouts. By selectively comparing these results, it was possible to detect the most serious errors before going into production, something many of the recent projects in Germany such as the Unemployment Payment II have failed to accomplish.

Keywords: Functional Testing, Black-Box Test, File Comparison, Selective Validation, Assertions

1. Project background

In 2002 the Austrian Chamber of Commerce – WKO – decided to finally replace it's 25 year old host applications, written in Assembler and PL/I, and move to a modern web-based environment. The goal was to get all of the software off the mainframe by no later than December 2004. The target environment selected was Microsoft's DotNet. The data was to be migrated from VSAM files and DB-2 databases to SQL-server relational databases. The programs were to be completely rewritten and provided with a Web XHTML-based front-end and an XML-based backend.

The effort involved in accomplishing this task was grossly underestimated, but that is not the issue here. The fact is that the requirement specifications and the database model were completed by the end of 2002 and the project for developing the major application – the due collection system – was launched in January 2003 with a targeted completion date of 7th January 2004. Originally there were 5 developers and 1 database specialist assigned to the project.

Progress was made and a lot of C# Code produced. However, by the fall of 2002 it became obvious that the project was in trouble. New requirements kept coming in and there were technical problems with the DotNet environment. There was a lack of version control and configuration management and the developers were not able to test because of the lack of data. The data migration from the host to the SQL server was still not functioning as late as November 2003.

At this point the software house Anecon was called in with 4 persons to strengthen the project. One set up the configuration management and introduced strict project management procedures. Another took over the data migration. The remaining two, including the author, began to set up a system test environment.

Of course, the project was not finished by January 7th 2004. The testing was only starting, but soon automated testing was in full progress. Within five months over 500 errors were detected and fixed. By July, 2004 the first version for the first Austrian state could be released. By the end of 2004 all of the Austrian states were using the system. The project was a year late and the costs were 2 ½ times more than what was originally budgeted, but it was successfully completed.

2. Risk oriented test approach

In view of the limited time available to test, an approach was chosen which would expose the most serious errors. In such a billing system the greatest risks lie in corrupted master data and incorrect invoices. The approach was to compare the data in the new databases after processing with the data in the old files after processing and to compare the contents of the new billing documents with the contents of the old bills.

The first comparison was intended to detect corrupted data, while the second comparison was directed toward catching incorrect invoices.

The comparison of the old and new results required very sophisticated tools, since the structure and contents of both the databases and the outputs were quite different. The old data was stored in 8 VSAM files on the mainframe. The new data was scattered throughout 124 relational tables on the SQL database server. The old invoices were printed centrally in Vienna and mailed out to the members throughout Austria. For the printing, a print file was created by the PL/1 programs on the host. The new billing data was given out in XML files which were then transmitted to the individual states to be printed locally.

3. Task of selective comparison

Since the old and the new data were so different, there was no alternative but to restrict the comparison to selected data items. In the database there were in every old file certain critical attributes such as the member number, the professional category, the payment rate and the payment interval. The task was to extract these data items from both the old files and the new databases and to store them in a common intermediate form which could be compared.

In the outputs there were also critical values such as the member address, the due date, the interest rate and the amount due. These values had to be extracted from the old print files and transcribed into XML files which could then be compared with the new XML files.

What is important, it had to be possible for the tester to select both individual data objects as well as individual attributes to be compared. Furthermore, it had to be possible to adjust the values of the attributes by adding values to or subtracting values from them or by outright overwriting them, since many codes were changed through the data migration.

4. Language for selective comparison

The solution was to design a test language for selecting and comparing data content. The language contains an IF expression for selecting data objects from either a CSV file or on XML file and then to compare selected data elements with each other. For this the data column names were used in the CSV files and the data element names in the XML files. The old data is identified with the prefix “old” whereas the new data is identified by the prefix “new”.

For comparing data items an assert expression is used which compares the new values against adjusted old values or against constants. This gives the tester some leeway in what is compared since even the old and new names can be different.

Later a “when… replace..” statement was introduced to replace certain values with another value based on the content of the record. This made it possible for the tester to adjust the data before comparing it.

```
FILE: LAENDER;
  when (old.STAATSID = "16") then skip;
  when (new.BUNDESLAND_ID = "9" then replace new.STAAT_ID by "0";
  if ( new.BUNDESLAND_ID = old.BUNDESLANDNR &
       new.STAAT_ID = old.STAATSID ! "10" )
    assert new.STAAT_ID = old.STAATSID ! "10",
    assert new.BUNDESLAND[1:4] = old.BUNDESLAND[1:4];
END;
```

5. Tools for selective comparison

To perform the selective validation, four tools were required, two on the mainframe and two on the PC-Server.

The task of the first tool on the mainframe was to extract a set of values from the VSAM files and to transpose them into a CSV file based on a list of parameters.

The task of the second tool on the mainframe was to extract selected values from the print file and to convert them over into an XML file based on an XML schema using an XSL pattern. The XSL pattern describes how the fields were to be located in the print file. The XML schema describes how the XML output file is to be structured. This tool required a lot of effort and had to be adjusted to every variation of

the print files, since every Austrian state required another printout format. This was the reason for writing the new outputs in XML files and leaving it up to the member states to print them out.

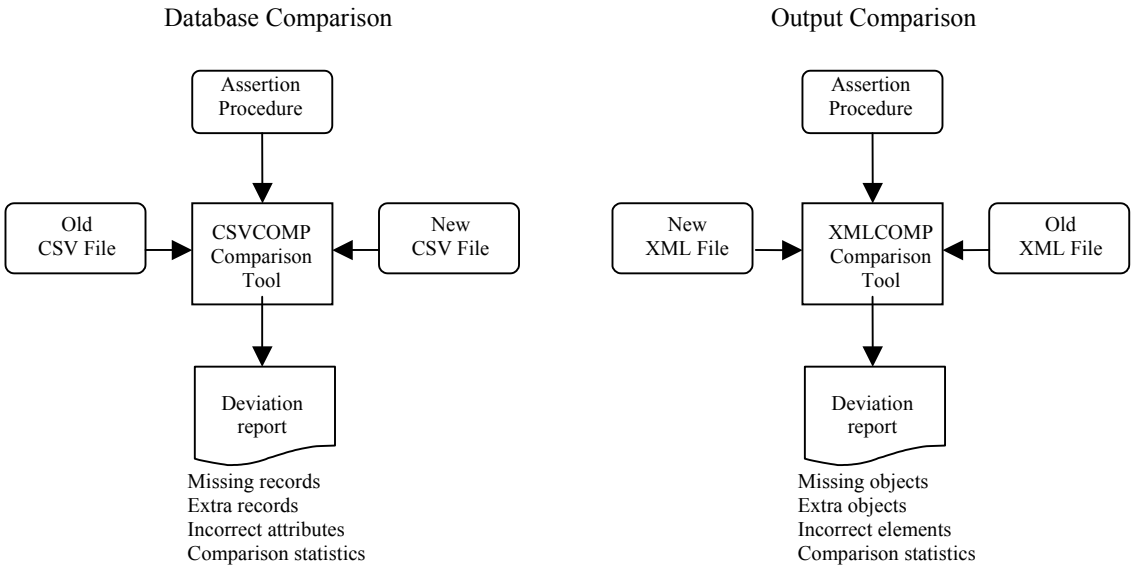

Figure 1: Comparison Tools

The tools on the PC were for performing the comparison. One tool interpreted a test assertion procedure and selectively compared CSV files. The other tool interpreted a test assertion procedure and selectively compared XML files. Both produced a similar report, giving the objects which were missing and those which did not match. In the case of non matching objects, the non matching items were listed out. Besides that, a statistic was provided on the comparison results.

In addition to these four tools a SQL query procedure was written for each new CSV file with a SELECT to pick out the selected values for comparison from the tables in which they were stored and a JOIN to merge them together in a single CSV file with a line for each tuple.

6. Selective validation process

To implement the selective validation test it was necessary to repeat the same process for every billing type for every Austrian state. The three billing types were

- the payment invoice,
- the reminder and
- the court notice.

The process involved some 10 steps.

Step 1: the billing was processed by the old system on the host
Step 2: the CSV files with selected data items were created from the VSAM files
Step 3: an XML file with selected data items was created from each invoice printout
Step 4: the host CSV and XML files were downloaded to the PC-server
Step 5: the same billing process was repeated the by new system in the DotNet environment
Step 6: the new CSV files were created by an SQL utility for selecting data attributes from several tables and joining them
Step 7: an assertion procedure was prepared for each CSV file to be compared
Step 8: each new CSV file was compared with the corresponding old CSV file and a validation report produced
Step 9: an assertion procedure was prepared for each XML file to be compared
Step 10: the new XML file was compared with the old XML file and a validation report produced

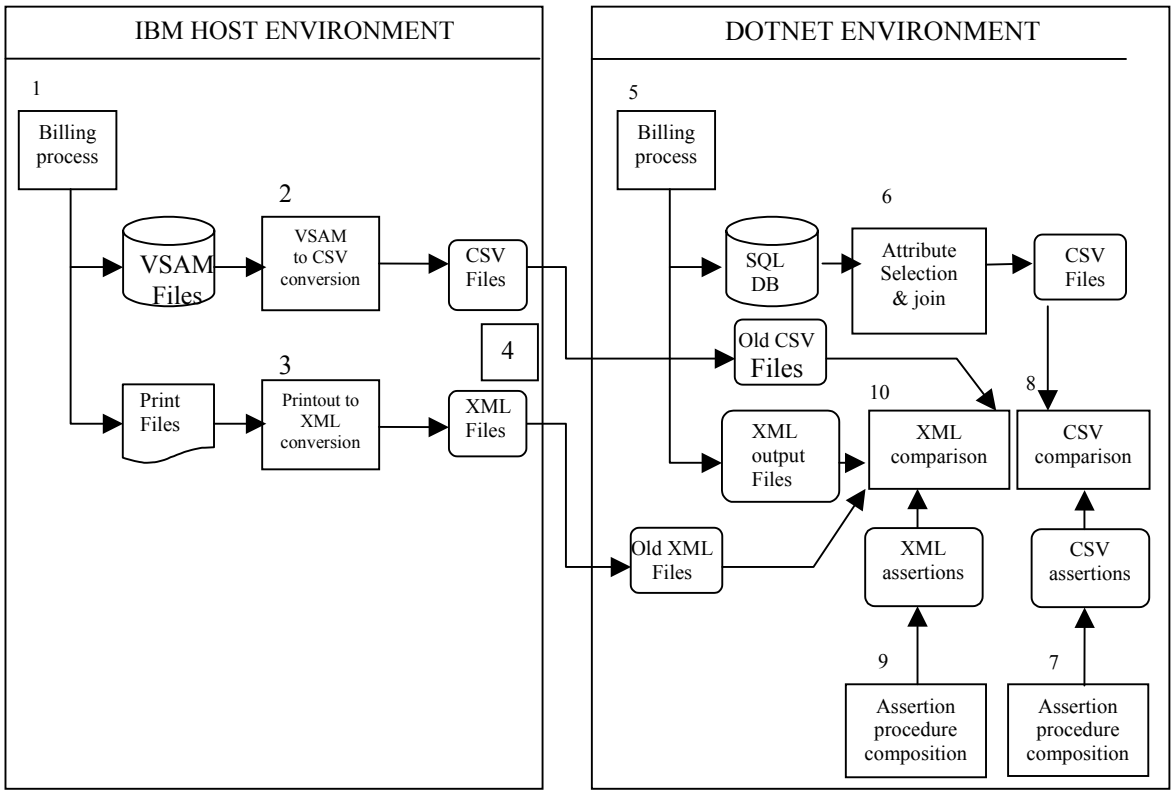

Figure 2: 10 Step Selective Validation Process

7. Selective validation benefits

The benefits of the selective validation process were threefold.

- For one, it was very fast due to the high rate of automation. A billing process could be tested within one day. The longest running step was the comparison step which in case of large files with more than 1 million records ran up to 3 hours.
- For the second, it was very cheap. The tools were developed by one person – the author – in 1 ½ months. The testing of a billing process could be made by one tester.
- For the third, it was very effective. The most critical errors such as
    - missing members
    - wrong addresses
    - incorrect categories
    - invalid account numbers
    - false amounts

  were detected and reported. Of the 527 errors reported in the 8 months system test, over 200 were a result of the data comparison. These were also the most critical. In this respect, the selective validation approach to testing attained its return on investment by exposing a large number of critical errors at a low cost in a short period of time.

Literaturhinweise

Sneed, H. “Testing Software for Internet Applications“, Software Focus, Vol. 1, No. 1, Sept. 2000, p. 15-22

Nguyen,H.Q.: “Testing Web-based Applications“, Software Testing & Quality Engineering, Vol. 2, No. 3, May 2000, p. 23-30

Fewster,M./Graham,D.: Software Test Automation, Addison-Wesley, New York, 1999