Paper 2491-2018

The Power of Testing! – An automated SAS® Enterprise Guide® System for Testing

Karine Désilets, Statistics Canada

ABSTRACT

Over the years, at Statistics Canada, the testing tasks have been distributed between the employees: unit tests are done by IT, the certification of the calculations by methodologists and subject-matter analysts approve the process flow and data quality. Most of those tasks can be, and should be, automated; especially the tasks related to unit tests and regression tests. With the help of the *Base SAS*®, SAS® GRID and SAS®/EG flow and a bit of Excel, this article presents an innovative way to automate testing. The main goal of this article is to use the power of SAS to avoid manual steps, maximise test coverage and minimise the amount of tests.

The article discusses the naïve heuristic-based methodology to automatically generate test cases for an application where XML files are used as input metadata. In large applications, it is impossible to cover the quality assurance of each path. The objective is to centralise tests on critical paths as well as to give some ideas on how to automate and tests non-critical paths. Secondly, the article focuses on how to do quality assurance testing from Excel based definition test cases. The article explains in detail how to architecture the different types of test: unit test, test cases, test scenarios and regression tests. SAS®/EG process flow, code examples and quick tips are given. The limitation of the methodology is also presented and discussed. The final emphasis is on future research on the SAS testing topic.

INTRODUCTION

How test cases should be created; where testing should be applied; who should do the testing; how often should the testing be done are most of the time hot topics when discussing testing strategies. However, the first questions that we should answer are: How to avoid manual steps? How to maximize test coverage? And, how to minimize the amount of tests?

This article is intended for managers, developers, testers and people interested in the testing topic with main interest in systems testing built with SAS® technologies. Some possibilities have already been explored in articles [1], [2], [3], [4], [5], [6] and [7]. As well, automatic comparisons of datasets have been explored in [8] and [9] and are the foundation of regression testing topic.

The first focus target strategic matter related to testing vision, governance and monitoring. Second, it describes some simple tactics in order to define tests in general. Then, the differentiation between unit test, test case and test scenarios is explained as well as regression testing. Lastly, the power of SAS is used in order to define and explain a testing system for large SAS applications using xml metadata file as input in conjuncture with SAS dataset.

As presented in the abstract, the article shows how to architect and record the different types of test in Microsoft® Excel spreadsheets, how to auto-generate simple and complex xml files from Excel spreadsheets, read the generated xml into a SAS dataset and output the xml file. The xml files produced are then read to test the SAS application and apply regression testing when needed. Ultimately, tests are automated, executed, validated, analyzed and visualized with SAS technologies. That is the power of SAS testing!

VISION AND ORIENTATION OF TEST ENVIRONMENTS

When testing systems or applications, the paradigm of developer versus tester (Figure 1) is often faced; most of the time that situation happens because management or clients wants to deploy the systems in production as fast as possible. Then, the testing phase is one of the phases that is shortened or mostly done with minimal steps. Secondly, creating a testing system or using one that already exists demands both time and a learning curve. The result of doing tests with minimal testing has a huge impact on the costs of the maintenance: In production; bugs need to be fixed.

Figure 1. Paradigm of Tester versus Developer[1]

In order to have a well-defined testing strategy, it is fundamental to first define the testing environments that are present in the system development phases. In the literature, it has minimally two environments; the development (DEV) and production (PROD) environments. Between these two standard environments, many other environments can be described and used. In that article (Figure 2), the Quality Assurance (QA), also called TEST environment is shown and the User Acceptance Tests (UAT) is represented. Other environments such as Integration (DEV_INT) could have been represented.

Over those testing environments, it needs a shared vision and robustness with short-term, mid-term and long-term actions in order to implement that vision. The governance main goal is to support the vision and, present tangible actions for the vision becoming reality. Mainly, the governance provides orientation, efficacy and efficiency. Lastly, the monitoring is done with performance indicators. The roles and responsibilities of each actor as well as what currently occurs versus the international industry standards should be one of the key actions to analyse and describe as found in ISO/IEC/IEEE Software Testing Guidelines [7].

As also stated by [7], software testing should have the commitment of the highest level of organizational management, be it the CEO, open-source steering committee or a department manager. With that commitment and the creation of both organizational Test Strategies and Test Policy, the testing performed in projects would be more effective and efficient. Those documentations would first provide guidelines about how to achieve all objectives stated in the Test Policy and would give guidelines for both safety-critical products and non-critical products.

Statistic Canada currently has mission-critical surveys that can impact stock market, PIB and other high-profile indicators. The Informatics Branch developing systems supporting those surveys follows testing protocols that are based on industry standards and best practices and also, does maintain the testing standards, conduct quality audits and coordinate specialized testing. At Statistic Canada we have a Test Policy, an Organizational Testing Strategy and a Testing Protocol.

TEST STRATEGIES

In application development, testing strategies are really important and should be well defined. Testing all the paths or combinations is impossible when developing applications [7]. Multiple techniques are often used in combination with others; we called them heterogeneous techniques. The first one I use is: extracting a percentage (%) of all the path combinations. That method is pretty simple and naïve however, we do not know if the most relevant paths are tested and if the tests are uniformly distributed among the paths. The second ones, to create tests, are heuristic based algorithms. This is where machine learning or artificial intelligence techniques fall. Finally, there are techniques based on knowledge: “I know the system, I know

[1] Image source : http://www.authorstream.com/Presentation/Tricon_Infotech-2170109-tester-vs-developer/

where development team made changes and then I know which tests to apply". Those types of tests are often done by experienced Programmers, Testers or experienced Team Leaders.

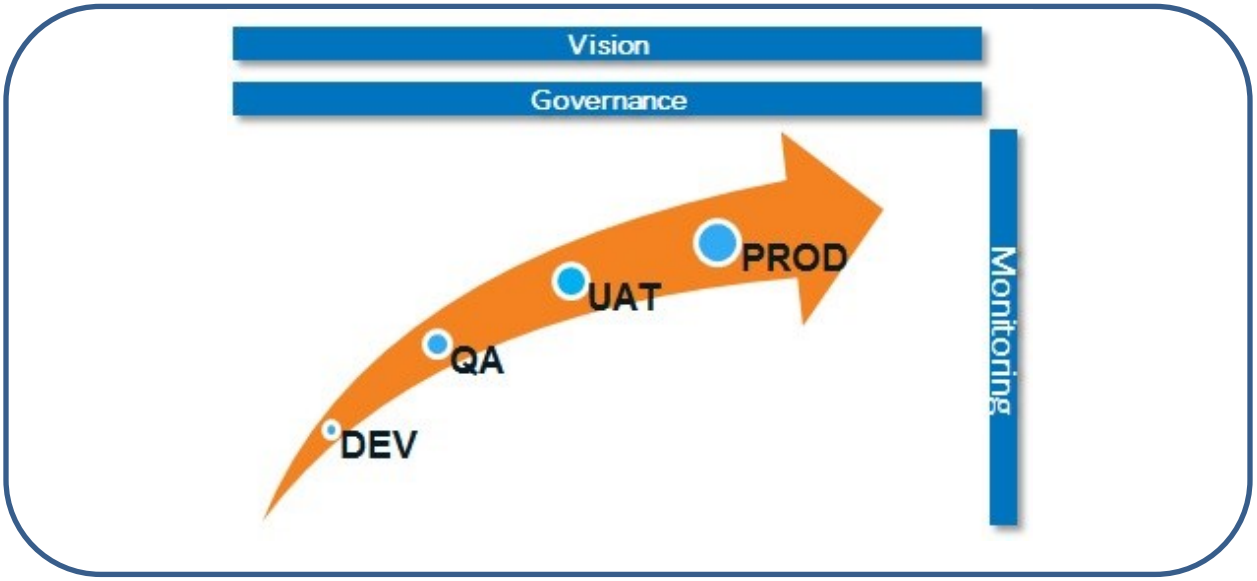


Figure 2. Vision, Governance and Monitoring of Testing Environments

All those testing techniques are good but there is reality; the triple constraint! Time versus Costs versus Resources! Do we have time? Do we have the resources? Do we have the money? Most of the time, the answer is: No! Then, we have to rely on metric as System's Maturity. A mature system has a formal and architecture testing system established. Also, the culture of the organisation as well as employees think about systems testing is important factors to consider.

Figure 3. Triple Constraints Reality

Lastly, the organisation developing the system needs to know who the testers are. In some organisations there are formal Tester Teams a.k.a. Testing Engineer or QA Engineer. In other organisations like Statistics Canada, most of the time the testing is spread among the employees (IT, methodologists and analysts) as people wear different hats when developing systems.

UNIT TEST, TEST CASES, TEST SCENARIOS AND REGRESSION TESTING

This article's emphasis is on four different types of testing: unit test, test cases, test scenarios and regression testing as found in [5], [6], [10] and [11]. Unit test is usually the first kind of testing that is done right after coding/debugging your application. Good practice is telling you to build unit tests as soon as code is ready – even before coding. Unit tests assess the smallest testable part of a system. Unit tests can be automated in big IDE like Visual Studio/Eclipse or, can also be manually driven as in SasUnit tool [4].

The next level consists of test cases; test cases are a set of variables/conditions under which a user will determine if the system responds correctly. It is fundamental to assign an ID to the test cases and an ID to

a test scenario, which consists of a collection of test cases. In a test case a lot of additional information [11] should be defined depending on your needs; as example: test case Id, expected result, expected data, actual result, author, description, summary, etc.

Finally, in order to confirm that test results of identical functionalities in different software versions show the same expected results (or a deviation is observed) in the new system, we use regression testing [10]. Regression testing can be used at the system behaviour; should my system still raise an error under those conditions? Or, at the data level; are my resulting data equal between versions? Automated regression testing will automatically compare dataset or system behaviours under different tests scenarios.

THE POWER OF TESTING! A SAS TEST TOOL TO TEST SAS APPLICATION

When the time comes to test SAS applications, there is no formal test tool. Some Unit Test tools are used by SAS developers as discussed in [1], [2], [3] and [4]. However, none of them discussed directly testing from information found in the xml input file with test cases, tests scenarios and/or regression testing. There lies the power of my testing system:

1. I automatically defined test cases directly in Excel from an algorithm; Manual definitions of tests can also be added in the final spreadsheet.
2. I auto-generated input xml file from the Excel test cases;
3. The SAS application(s) is(are) tested with the auto-generated input xml;
4. When possible, a regression test is automatically done.

First, let me provide some simple definitions:

- Let define X_i, $where\ 1 \leq i \leq p$, a finite set, non-empty, of xml parameters tags to test related to a specific functionality.
- Each specific X_i has a cardinal, $C(X_i)$, which is defined by the number of limit values $v_{ij}, where\ 1 \leq j \leq q$, of the tag parameters X_i.
- $n = \prod_{i=1}^{p} C(X_i)$ is the number of all possible input combinations to test.
- The limit values v_{ij} of each xml parameters X_i are defined by covering at minimum one or more potential values of the specific parameter X_i.
- The Cartesian product of each finite set X_i give all the possible combinations of parameters to test.
- The exhaustive test definition of the total set of X_i is often too large, the test coverage is then centralised on specific functionalities defined over xml parameters.
- The choice of parameters X_i to test is defined by the Tester and has to be carefully planned.

Thus, the key steps of the dynamic testing algorithm are the following:

1- Define the limit values of each X_i xml parameters to test in an Excel spreadsheet.

2- Produce the n test cases by doing all possible combinations of those limit values v_{ij}, (naïve algorithm). Save those test cases in Excel spreadsheet.

3- Automatically, Semi-Automatically or manually assign an *ExpectedResult* for each n test cases

- Succeeded: the application produces a result
- Failed: the application produces an error code

4- If there are some xml tags that can be static, defined them in Excel – their values will remain constant through all n tests

5- If there are semi-static tags, define them in Excel – all the n test cases defined previously will be tested for those k values tags

6- For each $n * k$ test cases, auto-generate the input xml and test the application.

- If succeeded – the application reproduces a result dataset
- If this is a new requirement:
  - Manually validate it, or;
  - Automatically validate it with another resulting dataset
- If a Regression test is possible, do it!
  1. Execute the xml on the current C_v and a previous system C_{v-a}, where $a < v$
  2. PROC COMPARE the two resulting datasets of each version, extract the automatic macro variables return code status *SYSERROR* and the PROC COMPARE return code *SYSINFO*
  3. If both automatic macro variables *SYSERROR* and *SYSINFO* equal zero (0) then Regression test is succeeded. If not equal to zero (0) manual analysis is needed.
- If the test produces an error code, extract the application error code
- Create accordingly two result datasets : ResultGood and ResultBad

Each step of that algorithm is explained in details in further sections.

GTAB-CENSUS AND GTAB EXAMPLE

In this article, the specific case of GTAB-Census (SAS system) is presented in the following sections in order to present the Testing Tool. GTAB-Census is a new Generalized Tabulation Tool (GTAB) for Census Data that is in development at Statistics Canada. GTAB, the former tool, is in the process to integrate Census specific calculations, confidentiality and integrating horizontal parallelism[2] of calculations in order to tabulate Census data in a timely fashion. However, the same principles presented in this article can be applied to other SAS applications driven by xml metadata.

Quickly, GTAB-Census needs two inputs: a SAS dataset with Census micro data and an xml metadata file containing all the specific parameters, calculations, dimensions, confidentiality and precision measures[3] needed to create tabulations. In Figure 4, a simple scenario of GTAB-Census is presented. The micro data file contains two dimensions (sex, age_group), a weight variable and an analysis variable, Income. The xml file specifies which calculations to apply on the micro data. The resulting SAS dataset table presents all the tabulated data. In that specific example, a weighted sum of Income is presented (no confidentiality algorithm applied and no precision measures considered).

[2] Horizontal parallelism differs from vertical one and occurs by creating parallel processes of the calculations on SAS dataset records. Vertical parallelism happens when creating parallel process with different groups of variables in order to recreate the whole calculations of the SAS dataset

[3] Precision measures are calculated from Bootstrap Method and include variance, standard error, coefficient of variation, confidence intervals and quality indicators

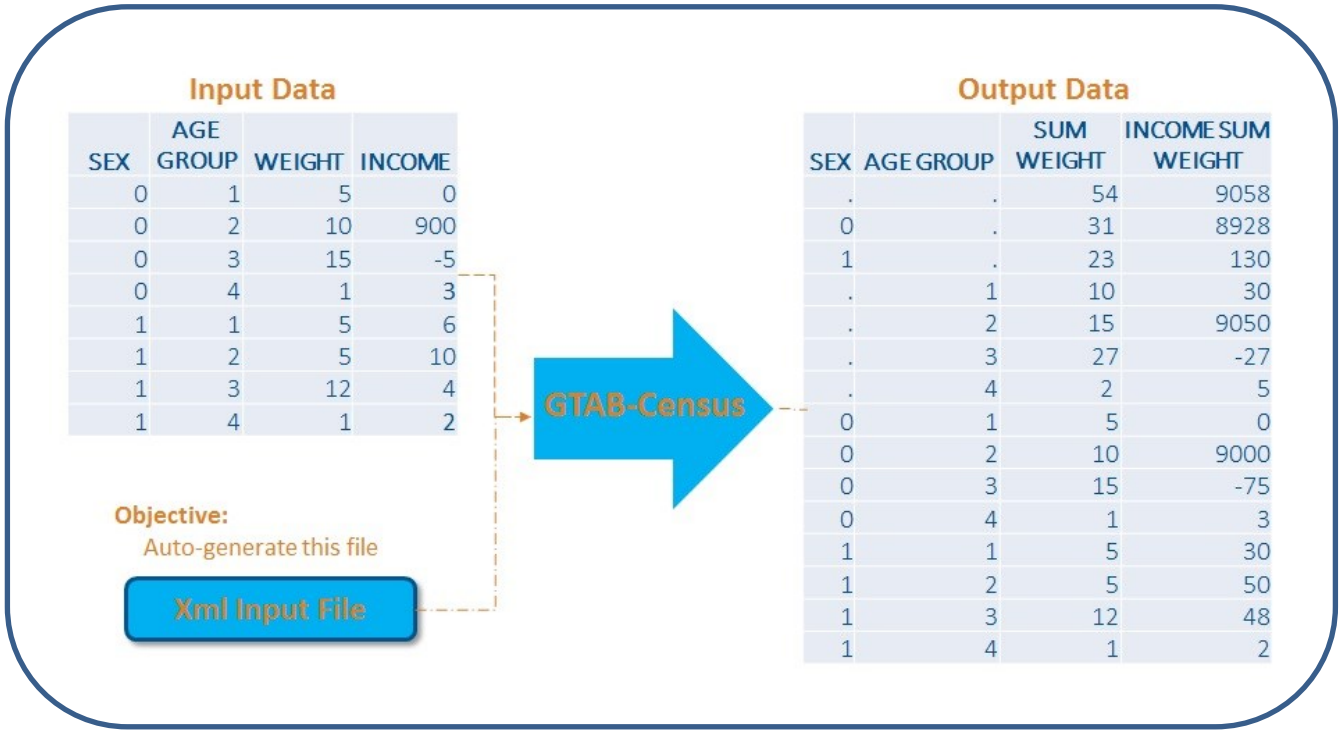


Figure 4. GTAB Sum and Weighted Sum Calculations Applied on Income Weighted Micro Data

As said, the main objective is to auto-generate the xml file containing the input metadata related to the application to test. As shown in Figure 5, the non-rectangular xml file is separated in three specific parts: *request_parameters*, *surveyweight* and *statistics*. The first part contains the name of the input micro data file (MyMicroData) and the name of the resulting file (MyTabulated). The second part contains the information related to *weight* variable; that part could be present or absent of the xml definition. The last part contains the calculations. Here, basic *sum* and *sum of the weights* are done on *Income* Variable.

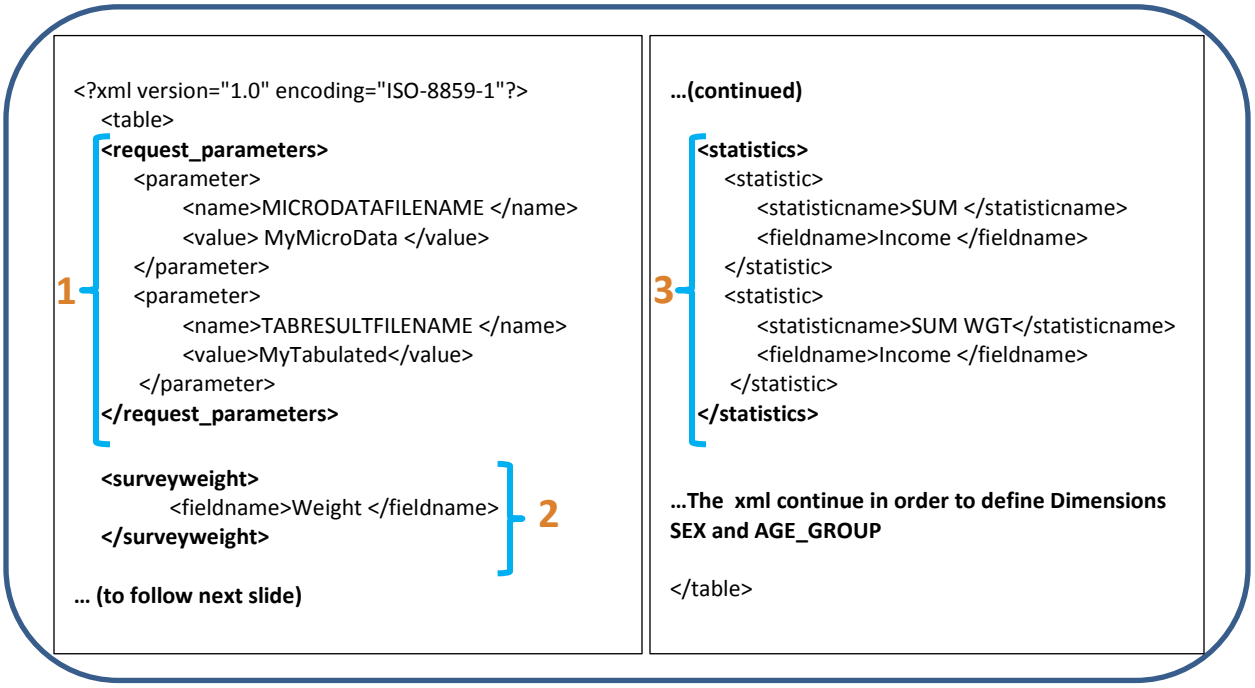


Figure 5. Ultra-Simplified Version of Xml Input File in GTAB

RECORDING TEST CASES DEFINITIONS IN EXCEL

Recording definitions in Microsoft Excel has been chosen because of the user-friendly matter of that application for manipulating data quickly. However, SAS datasets could also record definitions of tests. That being said, there are 3 types of definitions when a test would have been done on xml metadata. The first one are *static* definitions.

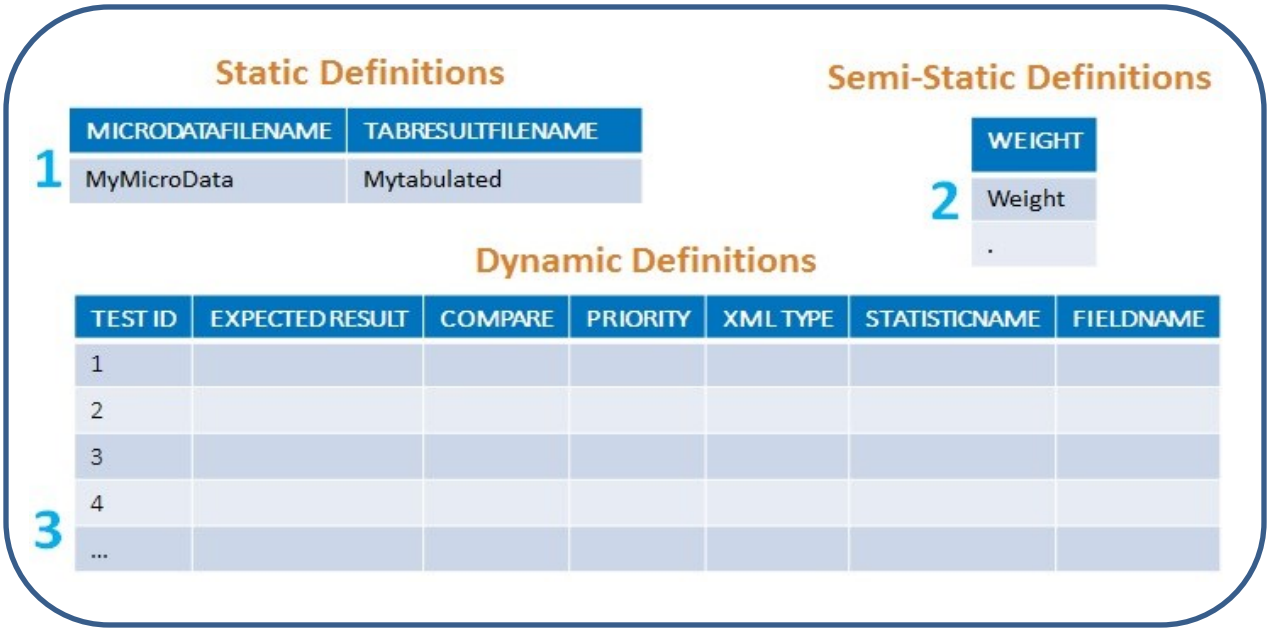


Figure 6. Step #1 Recording Tests

As shown in Figure 6, the *static* definitions record parameters that are not modified at all on every single test applied - as is the case with input microdata dataset and output result dataset. The second ones are *semi-statistic* definitions or facultative xml fields: test can be made with *weight* and *non-weight* values. And the last ones are *dynamic* definitions – the test definitions themselves.

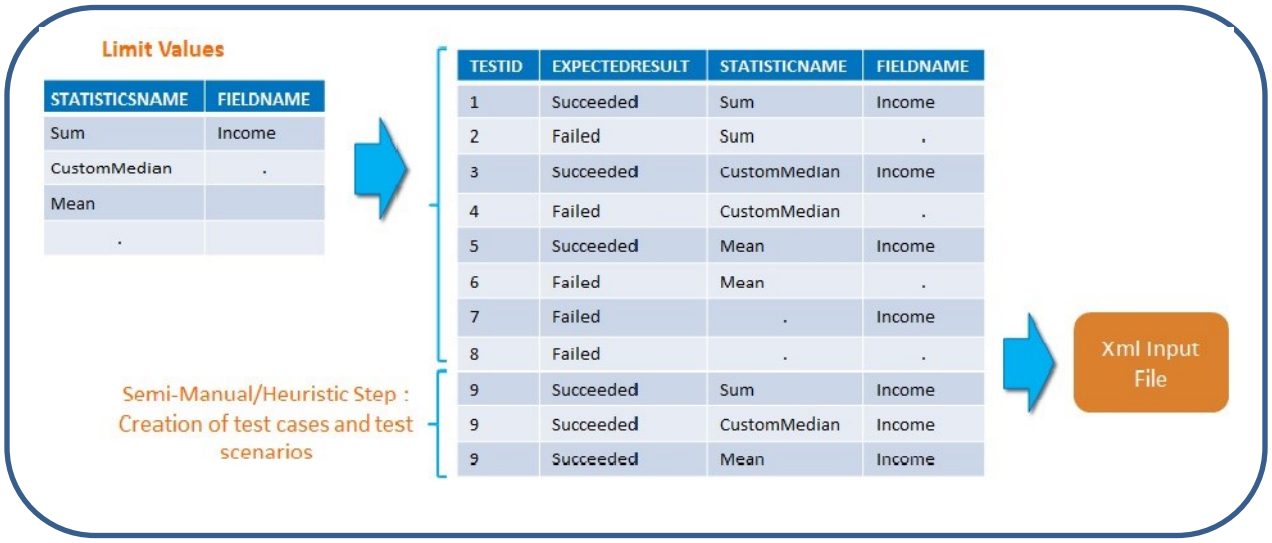


Figure 7. Step #2 Dynamic Definitions (Test Definitions)

The dynamic definitions are the tests themselves. Each field in the xml are represented in the spreadsheet by the variable *StatisticName* and *FieldName*. Limit values of each field are recorded in an Excel Table (Figure 6 - Limit Values spreadsheet). For example, *StatisticName* would take Statistics available in GTAB-

Census and empty (.) value. *FieldName* is the field where the statistic is applied: Income or empty(.). Those definitions could have *Invalid* value too. The naïve-based algorithm via the SAS macro ***%AllCombinations*** is then applied. All eight (8) test cases generated from limit values are shown in Figure 6. Some test scenarios can also be recorded as shown with TestID = 9. The xml then contains multiple definitions of calculations and idem in the final result dataset.

Supplementary definitions can be added in the test spreadsheet definition (Table 1). Some fields as *Regression* are needed to compare the resulting dataset with:

- a previous version of GTAB (*Regression* = GTAB);
- a previous version of the actual system used by clients (*Regression* = TPL);
- a new feature that methodologists needs to manually certify the results (*Regression* = Certification).

At this point, Census client could use the xml produced by IT team to start doing corroborations with the system they actually use (TPL)[4] and that we are in the process to replace.

TESTID	EXPECTEDRESULT	COMPARE	PRIORITY	XMLTYPE	STATISTICNAME	FIELDNAME
1	Succeeded	GTAB, TPL	1	automatic	Sum	Income
2	Failed	-	3	automatic	Sum	.
3	Succeeded	CERTIFICATION	2	automatic	CustomMedian	Income
4	Failed	-	3	automatic	CustomMedian	.
5	Succeeded	GTAB	1	automatic	Mean	Income
…	…	…	…	…	…	…
9	Succeeded	GTAB, TPL	1	automatic	Sum	Income
9	Succeeded	CERTIFICATION	1	automatic	CustomMedian	Income
9	Succeeded	GTAB	1	automatic	Mean	Income
10	Succeeded	GTAB	1	manual	-	-
11	Failed	-	1	manual	-	-

Table 1. Step #3 Supplementary Definitions

Another feature is "*priority*" variable. Some tests can have priorities – One is the higher and so on to the lower priority; the "*priority*" variable could then drive the testing approach. *Xml_type* can have two values, *automatic* or *manual*. *Manual* xml definition renders to xml created by clients when they test the application

[4] Some approaches have been made in order to validate actual results produce by GTABCensus with the result of the visualisation tool actually used by Census client: Beyond 20/20. As Beyond 20/20 produces a datacube and as GTABCensus produces that as well, that approach could be possible and, linking the results of both tools need further investigation. It could be easier to link output from GTabCensus and output from TPL in Beyond 20/20 format.

from the User Interface (UI). Each of those UI scenarios generate a specific xml and it can be saved under the rule: if the client approves the scenarios then it approves the xml. Certainly, those xml can be saved by the system and used in regression testing from one version to another. As well, "*notes*", "*description*", "*author*" can also be added as optional definitions and possibly many more as Testing System theory showed with [7] and [12] .

GENERATING XML TEST CASES

Generating xml is another hot topic, particularly in the case of GTabCensus xml: the xml is non-rectangular. This means that it cannot be outputted from SAS dataset with the xml .Map file. The simplest approach chosen is the reproduction of each section of the xml (Figure 5). It is done with SAS PUT statements in a dataset – see Appendix 4. That approach is SAS code driven by the xml structure. That could be improved. However, if the xml is rectangular, the xml can be directly output from SAS dataset and with the .Map file as presented in [13], [14] and [15].

LIMITATION

As said, it is impossible to test a software system exhaustively, thus testing is a sampling activity and is part of risk based-activity. Many techniques have been described in order to achieve testing and mitigate those risks. The actual prototype assigns priorities to some risks and then prioritizes the testing as described in Table 1. That way, the limitation of not testing exhaustively the software ensures that, at minimum, the risks with highest priorities are paid the highest attention [7].

The xml tags, structure and/or metadata's possible values are, sometimes, modified from one version to another particularly when multiple versions are iteratively released in QA environment. When those situations occur, all the test cases need to be regenerated from scratch; the automatic ones are quickly reproduced by modifying the generic program to construct the xml but; the manual ones need to be manually modified which can be sometimes painful and laborious. In order to limit changes to xml, GTab-Census team developed a compliant xml file to GTAB and stored the input metadata test cases in Microsoft Excel.

The basic ***%allCombinations*** algorithm could generate way to many combinations if it is not centralised on the proper limit values definitions: all the combinations are impossible to test. The tester should be aware of that limitation. However, when too many combinations are done, the tests can, in some cases, be combined in order to create test scenarios.

Generating xml metadata for a SAS dataset is a simple task when the xml is rectangular. In the case situation of a non-rectangular definition, it can become painful or impossible to generate xml from the xml Mapper. Some solutions have been given in [13], [14], [15] and [16]. For the moment, the non-rectangular GTAB-Census xml is generated with SAS PUT statements in a SAS dataset and copied in an .xml extension file. Further investigations have also been done; a methodology that is splitting the non-rectangular map (.map) file in multiple simple map definitions and then produce simple xml; Finally, all the simple xml definitions are merged together at the end. My team currently explores this solution.

When values included in a section of the xml depend on values on another section, it become tricky to create test cases. Most of the time it is easier to create simpler versions and process them separately than create much too complicated SAS programs to generate xml.

TESTING PROCESS FLOW

The testing process flow is pretty simple. For each test definition (TestID) in the spreadsheet, a unique xml is produced. That xml is used as input metadata to New System Version (GTAB-Census) and a Precedent System Version (GTAB). The two systems produce a result *Succeeded* when the system has executed without error code and *Failed* when it stopped and produced an error code. If both succeeded then, a regression testing is applied - a PROC COMPARE of resulting datasets is done. The comparison produced identical results? The New System Version succeeded regression testing if not; regression testing has failed. The two system results are compared to *ExpectedResult* variable. If one of the two systems failed, error codes are extracted, the *ExpectedResult* variable is compared and reports are produced (Table 2).

ExpectedResult	New System Result	Previous Version Result	Notes
Succeeded	Succeeded	Succeeded	PROC COMPARE is done. If new calculation then need Certification or comparison with TPL
Failed	Error Code	Error Code	Error Code Comparisons
Any other combinations of results			System Return Code Comparisons Need further analysis

Table 2. Testing Results Scenarios for Analysis Final Results

The two systems return Error Code and the *ExpectedResult* variable is assigned to *Failed* too; thus testing gives good results only if and only if the System Return Code have the same value. This kind of testing can lead to false positive: The expecting results mentioned *Failed* but both systems succeeded and produced results. Is it an error in both systems that need correction or is the *ExpectedResult* variable badly assigned for that test? Those cases can happen and need further analysis.

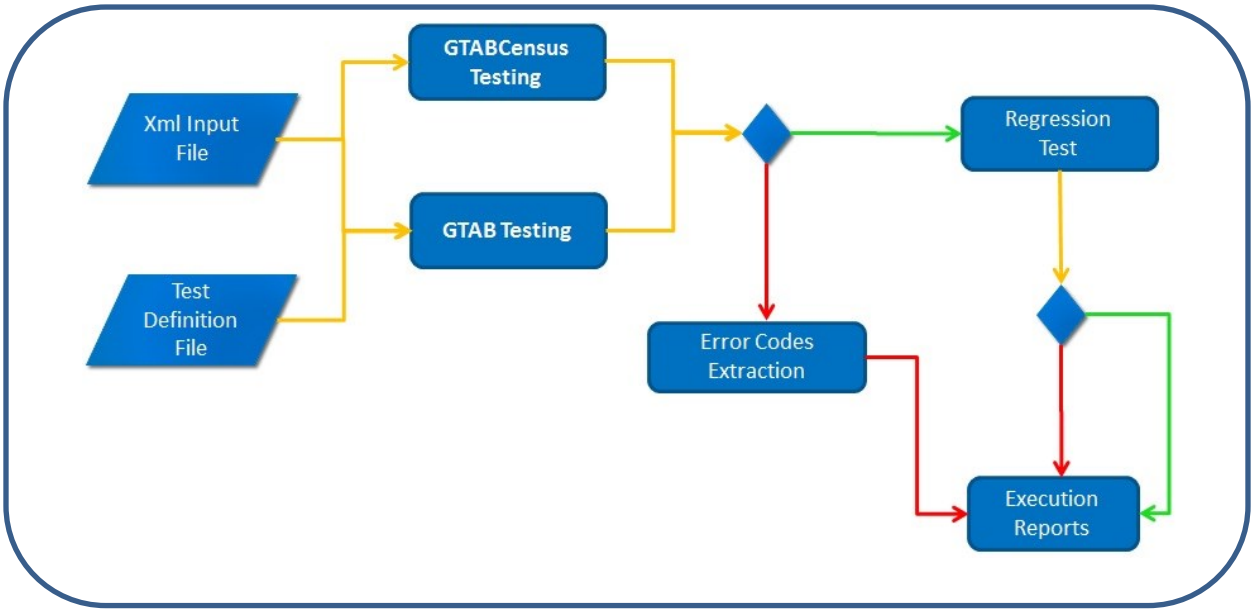


Figure 8. Testing Process Flow

Finally, we discovered with manual testing that some output from Regression testing did not produce good values in GTABCensus; it means that both system succeeded and the Regression Test was good but; the testing has not totally been covered first in GTAB and; if only comparisons testing is done, we could have missed some errors. However, as double comparisons are also done with TPL (current client System), we will probably see those errors too and narrow those cases. In fact, more a system is used and compared with other system results, more accurate the system will be.

SAS EG PROCESS FLOW

The SAS Enterprise Guide process flow is simple. As shown in Figure 9, there are four main process flows in the SAS EG project:

1. Autoexec
2. Produce Automated Test Cases
3. Create xml Files
4. Automated Testing

The first step creates the required libraries for the project. The second one creates the excel spreadsheet with all the test cases from the limit values of each xml tags. In that step, it is also possible to apply manual interventions, if needed. The SAS code about step 2 can be found in Appendix 2 and Appendix 3.

The third step produces the resulting xml files for all the test cases. That step could be included in the latest step as it currently saves all the different xmls in a folder. We might not need to preserve all of them in a folder; this could be an option in a future version. A simple example of the SAS code to generate an xml file automatically can be found in Appendix 4.

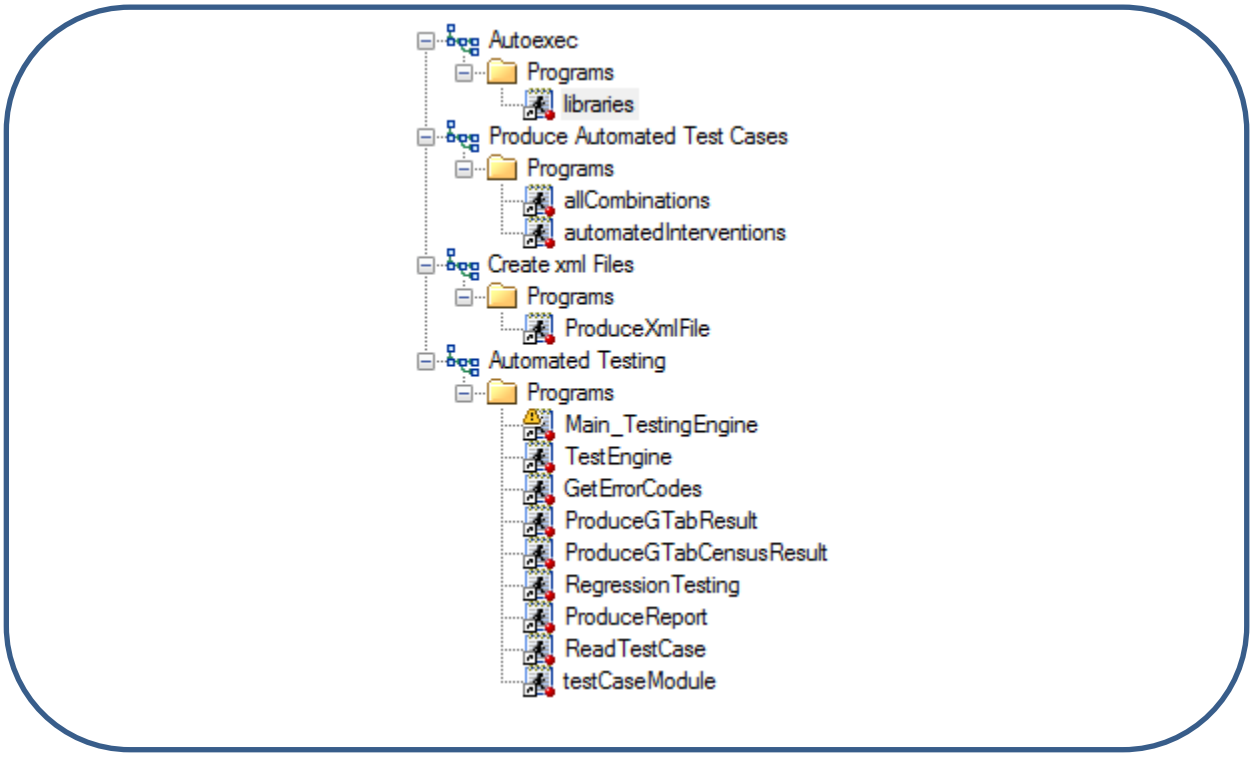


Figure 9. SAS EG Project Tree for Automated Testing Steps

The *Automated Testing* flow has a couple of SAS programs and modules; all of them are explained in Appendix 1. Both programs ProduceGTabResult.sas and ProduceGTabCensusResult.sas are exactly the same but, refers to two different versions of GTABCensus/GTAB (regression testing). However, those two modules could be generalized to one module and parameterized. Also, they both write the resulting datasets in a different folder in order to apply a final comparison of the output dataset with the program RegressionTesting.sas.

FINAL REPORTS OF TEST RESULTS

The final results of the *Automated Testing* flow are copied in two main datasets: *ResultGood* and *ResultBad*. As shown in Figure 10, the "Valid Results" of a regression test is decomposed into two segments:

1. For a specific *TestID*, if the *ExpectedResult* value has succeeded then *GTABTestResult* and *GTabCensusTestResult* return code have to be the same and must be blank value, *info* or *warning*. Also, *Regression* variable value must be equal to *Succeeded* (*SYSINFO* = 0 and *sysErr* = 0). If not, the test case needs to be analyzed.
2. For a specific *TestID*, if the *ExpectedResult* value is *Failed* then *GTABTestResult* and *GTabCensusTestResult* return code have to be the same and must equal to value *Error* or a numerical error code value. If the error code of both GTABCensus and GTAB is not the same, it needs further analysis.

Any other combinations of ExpectedResult, GTABTestResult and GTabCensusTestResult are considered to be Invalid Test Result. Secondly, If it is a new component, there is no regression test and an *ExpectedResult* of *Succeeded* imply manual authentication of the dataset when the *GTabCensusTestResult* is also blank value, *Info* or *Warning.* As well, an *ExpectedResult* of *Failed* implies the application will return an Error Code. The Error Code needs to be analyzed too.

For example, in TestID 1 to 9, the expected test results are *Succeeded* and GTAB and GTABCensus results are warnings then, a regression test have also been computed and they produced identical results: *succeeded*. In TestID 10 and 11, the expected result value is *Failed* and the two systems, GTABCensus and GTAB, produced error code (9025 and 9003). In that specific case, no regression has been applied.

1. Valid Results Dataset

RESULTGOOD

Filter and Sort | Query Builder | Where | Data | Describe | Graph | Analyze | Export

	testID	expectedResult	GTABTestResult	GTabCensusTestResult	Regression
1	1	Succeeded	Warning	Warning	Succeeded
2	2	Succeeded	Warning	Warning	Succeeded
3	3	Succeeded	Warning	Warning	Succeeded
4	4	Succeeded	Warning	Warning	Succeeded
5	5	Succeeded	Warning	Warning	Succeeded
6	6	Succeeded	Warning	Warning	Succeeded
7	7	Succeeded	Warning	Warning	Succeeded
8	8	Succeeded	Warning	Warning	Succeeded
9	9	Succeeded	Warning	Warning	Succeeded
10	10	Failed	1000	9025	N/A
11	11	Failed	1000	9003	N/A

2. Invalid Results Dataset

RESULTBAD

Filter and Sort | Query Builder | Where | Data | Describe | Graph | Analyze | Export

	testID	expectedResult	GTABTestResult	GTabCensusTestResult	Regression
1	36	Failed	Warning	Warning	Succeeded
2	80	Failed	Warning	Warning	Succeeded
3	89	Failed	Warning		Succeeded

Figure 10. Results Datasets and Reports

The Invalid results dataset for TestID 36, 80 and 89 shows that an *ExpectedResult* is *Failed* and the two systems, GTAB and GTABCensus, succeeded (warning or blank) and that a regression has been done and produced exactly the same results. This is a false positive! It needs analysis: is the *ExpectedResult* badly assigned? Are the two systems not executing properly under that xml and should produce an error code?

For the moment, the results of tests are in datasets but of course, reports and simple proc print are used for now. Reports involving former reports, SAS Visual Analytics or Power BI are in the plans and need further analysis.

RESULTS AND METRICS

I automated and processed many series of test cases when my team implemented some enhancements related to confidentiality functionalities to the GTabCensus Tabulation tool. In fact, my strategy to avoid confusion with the programmers and clients, in particular with the validations, was to define the limit values, expected results and Excel test cases before the programming started. That way, I was able to give well-defined expected results and avoid any misunderstanding.

As an example, a series of 800 xml test cases has been processed in less than 55 minutes on the Statistic Canada SAS GRID; including regression testing to a former version of GTAB. In order to give you other examples, I used that system and that strategy to test up to 5 totally different functionalities. Many other will be tested in the next Test phase in Spring 2018.

The purpose of the testing presented in Table 3 is mainly to ensure that the new functionalities are correctly implemented but also, to ensure that the final result dataset was not breaking out from version to version (regression test).

Major Functionality	Number of Tests Automatically Generated	Processing Time
1	432	34 minutes
2	4000	Not process yet
3	550	37 minutes
4	352	27 minutes
5	264	22 minutes

Table 3. Example of Test Cases Series Related to Confidentiality Functionalities

Finally, a shorter version of the testing tool has been implemented in order to process already existing complex test cases and achieve regression tests. That version process one xml file at a time but could be automated to process several xml files.

CONCLUSION

As you can see testing is not an exact science. The first goal of testing is to make sure that all the general paths of an application are tested from version to version ensuring consistency. Where it gets tricky is to test all the off-path scenarios; I have proposed different ways to minimize that risk by automating metadata test cases and recording production use of the system. The solutions I am proposing are not perfect but save testers a lot of time, ensure consistency and put in place a more rigorous testing method. We are basically going from a redo from scratch method to an automated system that can run thousands of tests within minutes.

The Statistics Canada SAS Grid does give a big help in terms of running tests in a timely fashion, don't get me wrong even without the grid it's better to run automated tests. It could take a day instead of taking weeks to redo everything by hand. There is no need to buy fancy COTS (Commercial Off-the-Shelf) testing tools, I believe that SAS gives all the tools that are needed! With Base SAS, the SAS Grid, SAS EG Flow and a bit of Microsoft excel, tests can be automated, executed, validated and visualized. The testing tools are important but the way it's being thought off and architect is even more important, you should take time to carefully plan the testing phase. In future articles, I would like to explore different topics related to testing like how we can use artificial intelligence or machine learning to improve testing. I would also like to add visual tools like SAS JMP or SAS Visual analytics in order to render the testing information in an easy way to understand.

With this article, a lot of exploration and standardisation has been done about automating your tests with SAS. Testing is always a hot topic that will never go away and where we will always have to battle for. In a fast pace world where delivery timelines are short: Users always ask for more! Users always want new products faster!

APPENDIX 1 - PROCESS FLOW, PROGRAMS AND THEIR OBJECTIVE

Step #	Step Name	Program	Objective
1	Autoexec	Libraries.sas	Assign libraries needed in the project
2	Produce Automated Test Cases	AllCombinations.sas	Create all the possible combinations from spreadsheet containing the limit values of each xml field definition SAS Code in Appendix 2
2	Produce Automated Test Cases	AutomatedInterventions.sas	Assign a value to ExpectedResult, Priority and Compare variables SAS Code in Appendix 3
3	Create xml file	ProduceXmlFile.sas	Create the xml files from the TestCase spreadsheet created in the above process flow SAS Code in Appendix 4
4	Automated Testing	Main_TestingEngine.sas	Main program to test all the test cases Program which create libnames and call %*TestEngine* macro.
4	Automated Testing	TestEngine.sas	Macro definition for testing the 2 systems
4	Automated Testing	ProduceGTabResult.sas	Process a test case case with GTAB system SAS Code in Appendix 5
4	Automated Testing	ProduceGTabCensusResult.sas	Process a test case case with GTabCensus system SAS Code in Appendix 5
4	Automated Testing	RegressionTesting.sas	Regression testing between GTabCensus and GTAB SAS Code in Appendix 6
4	Automated Testing	ProduceReport.sas	Produce final report over testing
4	Automated Testing	ReadTestCase.sas	Read a specific test case from the Excel spreadsheet
4	Automated Testing	TestCaseModule.sas	Extract error code from the different Error Code dataset when processing GTab or GTabCensus

APPENDIX 2 – SAS CODE EXAMPLE

Program: allCombinations.sas

```
proc import OUT=TESTSSCENARIOS
        DATAFILE="&teststatlevel"
        DBMS=xlsx REPLACE;
                        SHEET="allcases";
run;

data cleanTESTSSCENARIOS (drop= NOTES);
        set TESTSSCENARIOS;
run;

/*macro splitScenarioTable developed by Jun Li*/
%macro splitScenarioTable;
        %local dsid i nameVar rc;
        %let varList =;
        %let tableList =;
        %let dsid=%sysfunc(open(cleanTESTSSCENARIOS));
        %do i = 1 %to %sysfunc(attrn(&dsid., nvars));
                %let nameVar=%sysfunc(varname(&dsid,&i));
                %let varList=&varList &nameVar;
                %let tableList=&tableList level1_&nameVar.;
                data level1_&nameVar. (keep=&nameVar.);
                        set cleanTESTSSCENARIOS;
                        if not missing(&nameVar.) then do;
                        if &nameVar. in  (".") then &nameVar. = "";
                        output;
                        end;
                 run;
%end;
%let rc=%sysfunc(close(&dsid));

%mend splitScenarioTable;

%splitScenarioTable;

%let varList=%sysfunc(tranwrd(&varList., %str( ), %str(, )));
%let tableList=%sysfunc(tranwrd(&tableList., %str( ), %str(, )));

proc sql;
        create table allCombinations as
        select &varList., . as testresult
        from &tableList.;
quit;
```

APPENDIX 3 – SAS CODE EXAMPLE OF AUTOMATED INTERVENTIONS

Program: automatedInterventions.sas

Generating values for variables: ExpectedResult, Priority and Compare

```
data automatedInterventions;
        set allCombinations;
        testID = _n_;
        num = _n_;
        TestType = symget('TestType');
        if (TestType = 'stat1') then do; /*Assignation of ExpectedResult*/
                if upcase(statisticname) in  ("INVALID", ".", "") then ExpectedResult = 0;
                if upcase(fieldname) in  ("INVALID", ".", "") then ExpectedResult = 0;
                if  ExpectedResult  = .  then ExpectedResult = 1;
                end;
        end;

        if ExpectedResult = 1 then do;
                Priority = 1;
                compare = "GTAB, TPL";
        end;
        else do;
                Priority = 2;
                compare = "GTAB";
        end;

        xml_type = "AUTOMATIC" ;  /*if there is manual definition (xml_type = "manual"), they are added
manually in the final spreadsheet*/
run;


proc export OUTFILE = "&teststatlevelOut."
             data = automatedInterventions
            DBMS = xlsx REPLACE; sheet = "ALLCASE";
run;


/*manual interventions in Excel file can be done after that step:  assigning NOTES, or other manual
tests, interventions*/
```

APPENDIX 4 – EXAMPLE OF CREATING AN XML AUTOMATICALLY

Program: ProduceXmlFile.sas Creating the GTABCensus xml file – simple version

```
/********** WRITING XML FILE ************************/
%do i = 1 %to &nbLvl1 ;
        %do j = 1 %to &nbWeight;
                %let xmlOut = &xml/testcase_&i._&j..xml;
                filename outXML "&xmlOut";

                data _null_;
                        file outxml;
                        set parameters NOBS = Lst;
                        %let tab="  ";
                        Dim = dimensionname;
                        if _n_= 1 then do;
                        put '<?xml version="1.0" encoding="ISO-8859-1"?>';
                        put '<table>';
                                put &tab '<request_parameters>';
                        end;
                        else;
                                if DimLast = Dim then do;
                                        put &tab &tab '<parameter>';
                                        put &tab &tab &tab '<name>'parametername'</name>';
                                        put &tab &tab &tab '<value>'parametervalue'</value>';
                                        put &tab &tab '</parameter>';
                                end;

                        if _n_= lst then do;
                                put &tab '</request_parameters>';
                        end;
                run;

                data _null_;
                        file outxml mod;
                        set weight NOBS=Lst;
                        where testID = &j;
                        %let tab="  ";
                        Dim = dimensionname;
                        if fieldname ne '' then do;
                                put &tab '<surveyweight_fields>';
                                put &tab &tab '<fieldname>'fieldname'</fieldname>';
                                put &tab '</surveyweight_fields>';
                        end;
                run;
```

```
                        data _null_;
                                file outxml mod ;
                                set statisticsLevel1 NOBS=Lst;
                                where testID = &i;
                                by testID ;
                                %let tab="  ";
                                Dim = dimensionname;
                                confidrulewct = strip(confidrulewc);
                                TestType = symget('TestType');

                                put &tab '<statistics_level1>';
                                put &tab &tab '<statistics>';
                                put &tab &tab &tab '<statistic>';
                                put &tab &tab &tab  &tab '<statisticname>'statisticname'</statisticname>';
                                put &tab &tab &tab  &tab '<fieldname>'fieldname'</fieldname>';
                                put &tab &tab  &tab'</statistic>';
                                put &tab &tab '</statistics>';
                                put &tab '</statistics_level1>';
                        run;

                        /********** Some SAS PUT statements are needed to produce Dimension xml
                        statements for GTABCensus.  For simplicity, it is not showed in this example
                        ******/

                        data _null_;
                                file outxml mod ;
                                put '</table>';
                        run;
                %end;
        %end;
%mend ProduceXmlFile;


%ProduceXmlFile();
```

APPENDIX 5 - SAS CODE EXAMPLE TO EXECUTE A SAS MACRO CATALOG

Programs: ProduceGTabResult.sas and ProduceGTabCensusResult.sas

Here is a simplified version to call a sas macro catalog, execute it, save the resulting files in a folder and extract the errorCode – if any – at the end of the execution.

```
%macro ProduceGTabResult (xmlTestCase = , testID = );
        %let user_name = &sysuserid;
        %let version = 3.02.005;
        %let environment = DEV;
        %let GtabSystemPath = /sas/GTabCensus/ ;

        libname GTabLib ("/sas/GTabCensus/PROD/catalogs/v&version./") ACCESS=READONLY;

        %let GTabSystemFolder = _GTabSys;
        %let GTabPath = %sysfunc(dcreate(&GTabSystemFolder,%sysfunc(pathname(work))));
        libname _GTabLib ("&GTabPath");

        /*Copy GTab engine macro library and error format to work*/
        proc copy IN=GTabLib OUT=_GTabLib;
          select SASMACR   /MEMTYPE=CATALOG;
          select GTabFmts: /MEMTYPE=CATALOG;
        run;

        LIBNAME GTabLib CLEAR; /*Clear GTab macro library*/
        options mstored sasmstore = _GTabLib;

        libname gtabin "/sas/sasdata/proj/sastc/GTabCensus/Data/dataIn";
        libname gtabout "&folderTestPath./GTAB/";

        filename xref  "&xmlTestCase.";
        filename mylog "&folderTestPath./REPORTS/logGTAB&testID..log";

        %SocialSurveyGTab(confidFlag=1, xmlFileref=xref, debugmode=1);
        %getErrorFromGtab(OutDS= gtabout.getErrorFromGtab, macroRC = &_macroRC_.);

        %if (&_macroRC_. < 3 ) %then %do; /*Clear log if not error in GTAB*/
                dm log 'clear' output;
        %end;
        %sysmstoreclear;

        libname _GTabLib clear;
        filename Gtabsys "&GTabPath.";  /*delete GTab sasmacr catalogs from current work session*/

        data _null_;
           fname="sasmacr";
           rc=filename(fname,"&GTabPath/sasmacr.sas7bcat");
           if rc = 0 and fexist(fname) then  rc=fdelete(fname);
           rc=filename(fname);
        run;
        %symdel GTabPath;

        proc printto; run;/*log to windows log*/

%mend ProduceGTabResult;
```

APPENDIX 6 - SAS CODE EXAMPLE FOR REGRESSION TESTING

Program: RegressionTesting.sas

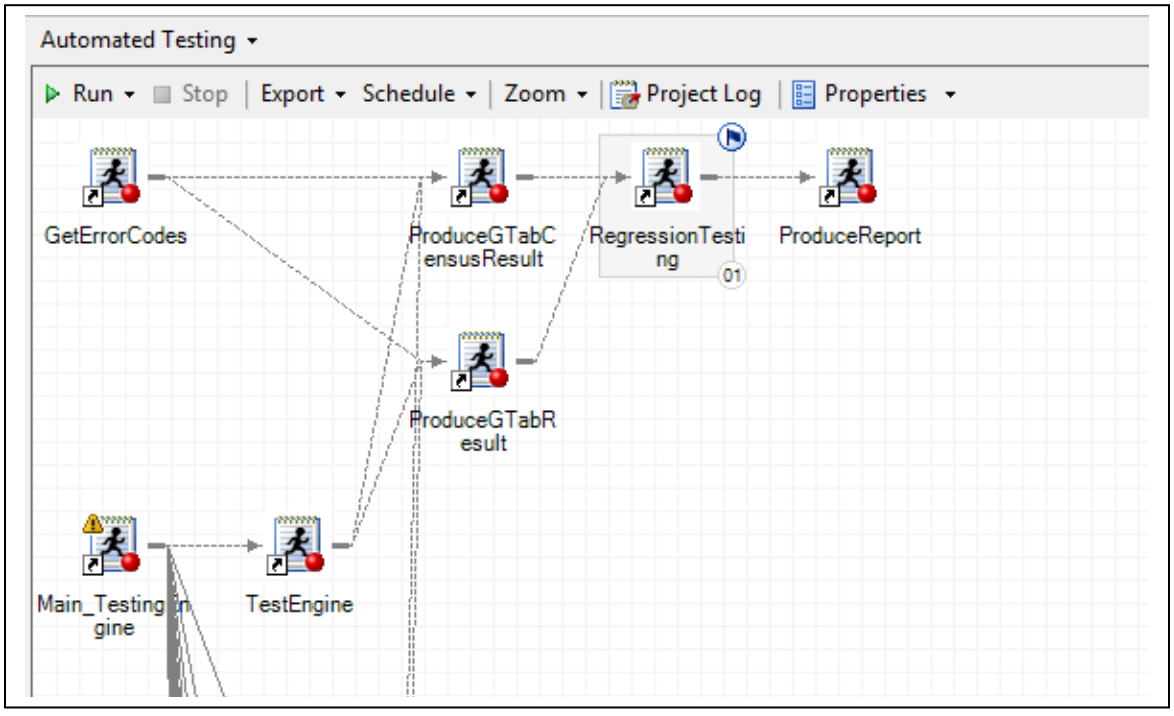
```
%macro RegressionTesting (gtabFile= , gtabCensusfile=,  CompResult=);

        proc compare data = &gtabFile. compare = &gtabCensusfile. out = &CompResult. ;
        run;

        %let RegressionErrCode =&syserr;
        %let RegressionSysinfo =&sysinfo;


        %put **********************************;
        %put RegressionStandErrCode =&syserr;
        %put RegressionStandSysinfo =&sysinfo;
        %put **********************************;
%mend RegressionTesting;
```

AUTOMATED TESTING – PROCESS FLOW



REFERENCES

[1] A. Ratcliffe, "Automated Testing of Your SAS® Code, and Collation of Results (Using Hash Tables)," in *SAS Global Forum*, San Francisco, 2013.

[2] D. A. Scocca, "Automated Unit Testing for SAS®," in *SAS Global forum*, San Antonio, 2008.

[3] "SAS Community Wiki - FUTS Framework for Unit Testing SAS," SAS Institute, [Online]. Available: http://www.sascommunity.org/wiki/FUTS__Framework_for_Unit_Testing_SAS. [Accessed 2017].

[4] A. Mangold and P. Warnat, "Automatic Unit Testing of SAS Programs with SASUnit," in *PhUSE 1*, Manchester, 2008.

[5] Wikipedia, "Wikipedia - Unit testing," [Online]. Available: https://en.wikipedia.org/wiki/Unit_testing. [Accessed October 2017].

[6] TechTarget, "TechTarget - Unit Testing," January 2017. [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/unit-testing. [Accessed October 2017].

[7] International Standard ISO/IEC/IEEE, Software and Systems Engineering – Software Testing; 29119-1, 29119-2, 29119-3, 29119-4, 29119-5, ISO/IEC/IEEE, 2013.

[8] H. Joseph and C. Margaret, "Deciphering PROC COMPARE Codes: The Use of the bAND Function," in *SAS Global Forum*, 2012.

[9] U. Gautam and J. Roberts, "Obtaining an Automated Summary of PROC COMPARE Results: &SYSINFO and VB Script," in *PharmaSUG 2017 – Paper AD06*, 2017.

[10] Wikipedia, "Wikipedia - Regression Testing," [Online]. Available: https://en.wikipedia.org/wiki/Regression_testing. [Accessed 01 2018].

[11] Wikipedia, "Wikipedia - Test Case," [Online]. Available: https://en.wikipedia.org/wiki/Test_case. [Accessed 01 2018].

[12] Software Testing Class, "What is the difference between test cases versus test scenarios," softwaretestingclass, [Online]. Available: http://www.softwaretestingclass.com/what-is-difference-between-test-cases-vs-test-scenarios/. [Accessed October 2017].

[13] M. Palmer, "XML in the DATA Step," in *SUGI*, Montréal, 2004.

[14] S. N. Jack, "XML Primer for SAS® Programmers," in *SUGI*, Philadelphia, USA, 2005.

[15] M. Cisternas and R. Cisternas, "Reading and Writing XML files from SAS®," in *SUGI*, Montréal, 2004.

[16] SAS Institute Inc., "SAS Support XML Tip Sheet," 2011. [Online]. Available: https://support.sas.com/rnd/base/xmlengine/XMLtipsheet.pdf.

[17] C. W. Schacherer, "The SAS® Programmer's Guide to XML and Web Services," in *MWSUG*, Minneapolis, 2012.

ACKNOWLEDGMENTS

I would like to first thank people of SAS Tech Center and people of Generalized Solutions at Statistics Canada: it is a pleasure to feel that I am part of both teams. Thanks All! Thanks also to my management colleagues: Etienne, Yves and Pierre. A Special thanks to Jun Li, my colleague and to Joel Orr and his wife who revised the paper. Finally, I would like to thank my husband and my kids. You always encourage me to write on ideas I have. Without your encouragements and free time you provided me at home to write, it won't have been possible!

RECOMMENDED READING

- *International Standard ISO/IEC/IEEE; Software and Systems Engineering – Software Testing; 29119-1, 29119-2, 29119-3, 29119-4, 29119-5.*
- *Informatics Branch, Statistics Canada; Testing Policy, Testing and Quality Assurance Initiative, Version 1.2, IB-TQA-0010.*
- *Informatics Branch, Statistics Canada; Organizational Testing Strategy, Testing and Quality Assurance Initiative, Version 0.8, IB-TQA-0011.*
- *Informatics Branch, Statistics Canada; Organizational Testing Protocol, Testing and Quality Assurance Initiative, version 0.6, IB-TQA-9999.*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Karine Désilets
Technical Advisor and Team Leader
Statistics Canada, Government of Canada
613-862-4220
karine.desilets@canada.ca
www.statcan.gc.ca