International Conference on Computational Science and Information Management (ICoCSIM)
Vol. 1 (2012) 233-238

Lessons Learned from the Implementation of xGTWay families of Unit Testing Tool

Maryam 'Afaf Abdul Karim[a], Rozmie Razif Othman[b], Kamal Z. Zamli[c]*

[a]*School of Electrical and Elecronics, Universiti Sains Malaysia, Engineering Campus, 14300 Nibong Tebal, Penang, Malaysia*
[b]*School of Computer and Communication, Universiti Malaysia Perlis, PO Box 77, d/a Pejabat Pos Besar, 01007 Kangar, Perlis, Malaysia*
[c]*Faculty of Computer Systems and Software Engineering, Universiti Malaysia Pahang, Lebuh Tun Razak, 26300 Kuantan, Pahang, Malaysia*

Abstract

Unit testing can be a long and repetitive process typically requiring laborious efforts. In an effort to alleviate such a burden, we have developed a set of evolving automation tools, known as xGTWay, as part of our research and development projects. Over the last 7 years, xGTWay has evolved through numerous evolutions yet maintaining the same objectives as its predecessor (i.e.to automate unit testing as much as possible). In this paper, the entire generations of xGTWay are presented along with the comparative analysis between them.

Keywords: Unit Testing; Test Automation, xGTWay

1. Introduction

Unit testing involves testing a specific component or function within software system [1, 2]. The boundary that defines the unit to be tested can be subjective and is up to the tester to identify. Typically, unit testing focuses on a specific portion of codes that make up the system of interest. To ensure that unit testing usefully captures unwanted defects; testers typically target to cover all lines of code in the implementation. Here, testers are required to develop many test cases where each of which represents specific condition or scenario that is relevant to the code being tested. On top of that, testers will have to execute these test cases and analyze their results for conformance analysis based on some defined test oracles. If number of test cases in the range of hundreds, manual processes to write, execute, and analyze test cases can be painstakingly dull and lengthy [3, 4]. Furthermore, the need to retest codes due to design or code changes can also be unattractive and burdensome activity [5].

To alleviate the aforementioned problems, testers often employ automation tools. With automation tools, the entire testing process can be completely automated, hence, lighten the burden on testers from mundane and labor intensive chores. Enabling seamless automation as our main aim, we have developed xGTWay as part of our research and development project. In the past 7 years, xGTWay has evolved into 5 families of implementations from SFIT [6], JTst [7], G2Way [8], GTWay [9], and xGTWay respectively. In this paper, the detailed description of each family of xGTWay will be elaborated. In doing so, a summary of feature comparison will be made accordingly.

2. Evolution of xGTWay

This section highlights the evolution of xGTWay families by highlighting their similarities and differences in order to reflect on some of the lessons learned.

2.1. SFIT

The earliest family member, Software Fault Injection Tool (SFIT) [6, 10-12] uses Java technology to automate unit testing and enables up to 2500 test cases to be executed automatically. Implementation wise, SFIT relies on the users to provide the input test values and Java Reflection Application Programming Interface (API) to be able to construct the test drivers automatically. The use of Reflection APIs enables the program to access the internal structure of classes to be tested. Being able to access the internal class information allows SFIT to executetest cases automatically without user intervention, thereby alleviating the problem concerning the effort required in creating test drivers and the test framework.

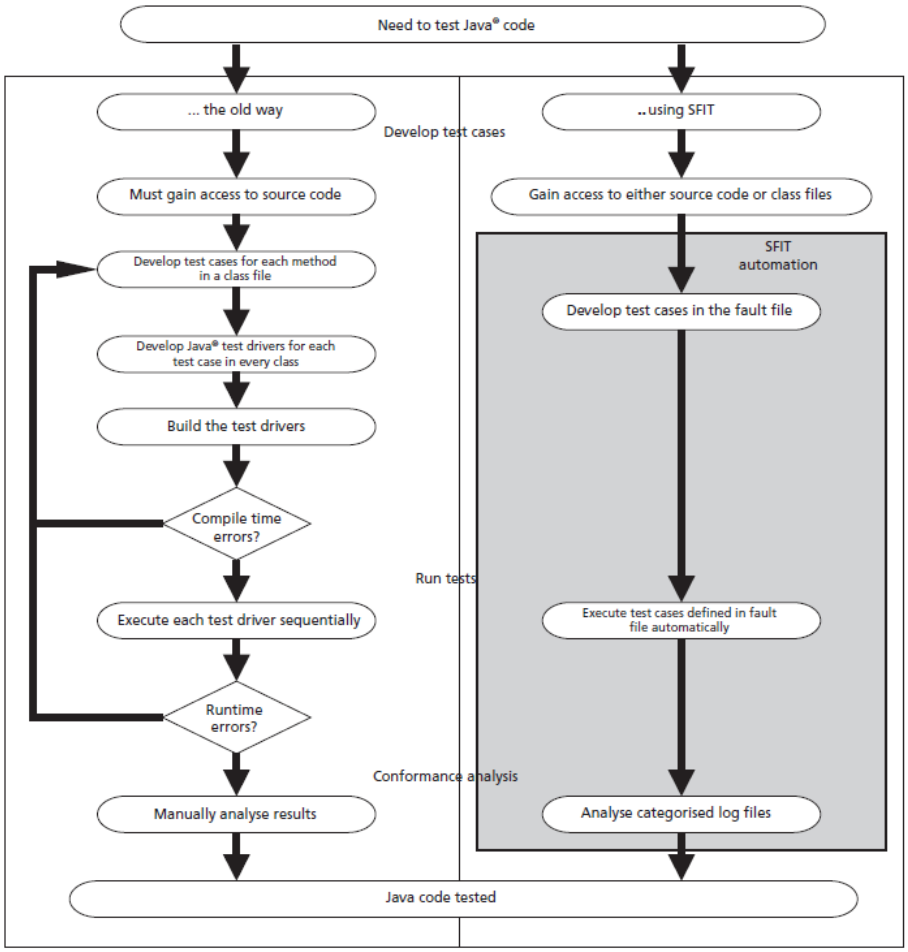

Fig. 1. Traditional Testing Method Compared to SFIT [11]

The architecture of the program consists of a few classes that are specialized in performing specific functions. There are five mentioned components of SFIT: Class Inspector, Fault Setting, Loader and Generator, Fault Injector and Log. Class inspector is used for interrogating the unit to be tested. Fault setting captures the information related to the test cases to be used for testing. The program uses a file with special formats called fault file in order to understand the requirement of testing that has to be done. The loader and generator are used for executing the test cases and generating intermediate files, which are to be utilized during the testing process. Finally, fault injector is for injecting the tests or faults that have been specified in the fault file and log is used for saving information pertaining to the test automation that has been executed.

Comparative study in the literature demonstrates that SFIT is capable of automating test drivers and execute them within acceptable speed. The software is able to simplify the traditional testing method shown in Fig. 1, that is, by eliminating multiple manual steps required which includes creating the test drivers and iterating the actual test case execution.

2.2. JTst

Following SFIT, the next generation of test automation tool, called JTst[7, 13, 14], was created. Its inception was due to the limitation of the automation capability seen in SFIT. Based on the same principle, the new software addresses the same unit test automation concerns as its predecessor. Unlike SFIT, JTst has additional features including automated test input data generation and the ability to execute test cases concurrently. JTst was designed with the purpose of generating test data using combinatorial approach made up of greedy algorithms and t-way combinations. The test data generation was implemented using the combinatorial approach with two possibilities. The first method considers varying one sensitivity variable for test generation based on specific parameters. The second methodvariesmultiple sensitivity variablesbased on any given parameters.

Table 1. SFIT versus JTst

SFIT	JTst
Class Inspector	Class Inspector
Fault Setting	Text Editor
Loader and Generator	Automated Loader
Log	Data Logger

JTst consists of five components: Class Inspector, Test Editor, Test Combinator, Automated Loader and Data Logger. Many componets of JTst are reuse from SFIT, however the list of components differ with the addition of test combinator and automated loader for supporting combinatorial approach and concurrent test execution. Table 1 lists the components that are common between the two tools. The function of test combinator is to generate combination of test inputs based on the user provided test case, which represents the base test cases that can be sourced from the program specification or previous data that causes the program to fail. The algorithm used is based on the modified greedy algorithm and the sensitivity variable can be a combination of two or more input parameters.

JTst was adopted to evaluate the robustness of Jada, which is a Java implementation of a parallel programming model called Linda. From the experiment done, JTst is able to test Jada tuple space operations even in the absence of the source codes. This is doable due to the use of Reflection API in retrieving the structure of the classes being tested. Applicability of JTst in testing environment is also backed up with the observation of some of the useful features that it offers such as generating test drivers and concurrent test executions. Upon test completion, a few loopholes were found in Jada when it is supplied with unsupported or out of range input values.

2.3. G2Way

Like JTst, .G2Way [8] provide a mechanism to generate the combination of test cases to be executed and concurrent execution of test cases. Nonetheless, the strategy used for test data generation in G2Way is different than that of JTst. Specifically, G2Way allows pairwise values of combination to be produced by combining all pair values for each input parameter.

G2Way is made up of four components: Parser Algorithm, Pair Generation Algorithm, Backtracking Algorithm and Executor Algorithm. The overall flow of test execution is similar to its predecessor. First, the base test data is provided in the form of fault files and parsed by the parser algorithm. The pair generation algorithm uses the values from the fault files to generate combination of test data for the test execution. The algorithm iterates through the number of possible indexes based on the input parameter value and analyzes its binary representation to find those with only two bits having the value of 1. Based on the result, possible pairwise values combination for each parameter can be produced by recombining all pair values of each parameter.

Next, the backtracking algorithm is employed to produce a complete combination of test data based on the results obtained from the pairwise generation process. The executor runs the test automation after all possible combinations have been collected. Parallel execution is possible in G2Way by utilizing multiple threads. Comparative experiments were undertaken to compare the performance of G2Way against JTst using a web-based configuration example consisting of 4 parameters with each of them having 3 values. Based on the results, G2Way efficiently generated 10 possible combinations as compared to JTst with redundant 27 combinations.

2.4. GTWay

GTWay [9, 15] represents a natural progression from G2Way. Instead of supporting only pairwise test generation, GTWay addresses the general t-way combinations, where t represents the general interaction strength. During the t-way pair generation process, the amount of indexes with binary bit set to 1 is counted and those that have the same count as the selected t parameter are chosen. For example, for 3-way combinations where t is equal to 3, indexes 7, 11, 13 and 14 with their binary representations 0111, 1011, 1101 and 1110 respectively, are chosen. Although earlier research shows favorable results with the increasing interaction strength, parameter t, the method is not efficient when the number of parameter coverage is very large[16]. When the amount of input parameter increases, there is a high chance that it will face combinatorial explosion problem since the number of t-way test sets increases exponentially.

Therefore, a separate research focusing only on test data generation was conducted. In this study, Modified In Parameter Order General (MIPOG) [16-18]was proposed, which is an extension of In Parameter Order General[19, 20] (IPOG) that allows both horizontal and vertical parameters extensions. With this approach, smaller sized test suites with acceptable execution time were generated compared to other methods, especially IPOG.

2.5. xGTWay

xGTWay represents the most recent family of implementations. Notably, the ability to generate test data automatically does not exist in xGTway Similar to GTWay, the users have to provide the test data as the input parameters of the test executor in the form of fault files. xGTway removes the burden from the tester to have to design the framework of each of the unit test. Instead, testers are only required to provide the test values that they would want to check through fault files with specific syntax. The program automatically generates the structure of the test case in Java language after analyzing the specified functions to be tested. Then, the execution of test cases is automated. The result of the test is logged to a file for the user to analyze after the testing completes. The process is supported by four components: Class Inspector, Test Editor, Test Executor and Data Logger.

Unlike all of its predecessors, xGTWay is capable of resuming execution from the last saved state. With the fault tolerant-like characteristic, if the test execution is accidentally aborted or voluntarily halted, the entire automated process does not have to be restarted from the beginning, thereby, wasting time and efforts that have been put earlier. Instead, the test automation can continue its execution on the last saved state known as a checkpoint. In the current design, users can specify the checkpoint interval either statically or dynamically.

3. Comparison amongst xGTWay Families of Implementations

Based on the discussion on earlier sections, this section summarizes the features of each of the xGTWay families of implementations. To facilitate discussion, the main characteristics that are common have been identified including the abilities to:

- Generate test data
- Execute test cases automatically
- Support concurrent execution of test cases
- Recover and resume test execution from interruption

Table 2. Feature Comparisons between xGTWay Families of Implementations

SFIT	SFIT	JTst	G2Way	GTWay	xGTWay
Generate test data	X	√ (based on parameter sensitivity)	√ (based on pairwise interaction)	√ (based on t-way interaction)	X
Execute test cases automatically	√	√	√	√	√
Support concurrent execution of test cases	X	√	√	√	Not fully supported
Recover and resume test execution from interruption	X	X	X	X	X

√ = Applicable , X = Not Supported

Referring to Table 2, no single implementation has all the desired features. Here, each implementation gives focus on certain aspect of the features of interests. Notably, xGTWay has not fully supported concurrent test execution. In the current version, concurrent test execution can have unwanted side effect (i.e. inconsistent state of execution) owing to its ability to support recovery and resumption.

A more subtle difference between xGTWay families lies in its process flow as shown in Fig. 2. Here, JTst, G2Way and GTWay are grouped together since all three of them have the automated test generation capability. On the other hand, xGTWay has another alternative path to automate the test execution through its resumption facility, which is to resume test execution from where it stopped. All implementations merge toward same point within the automation process, that is, during the automated test execution activity.

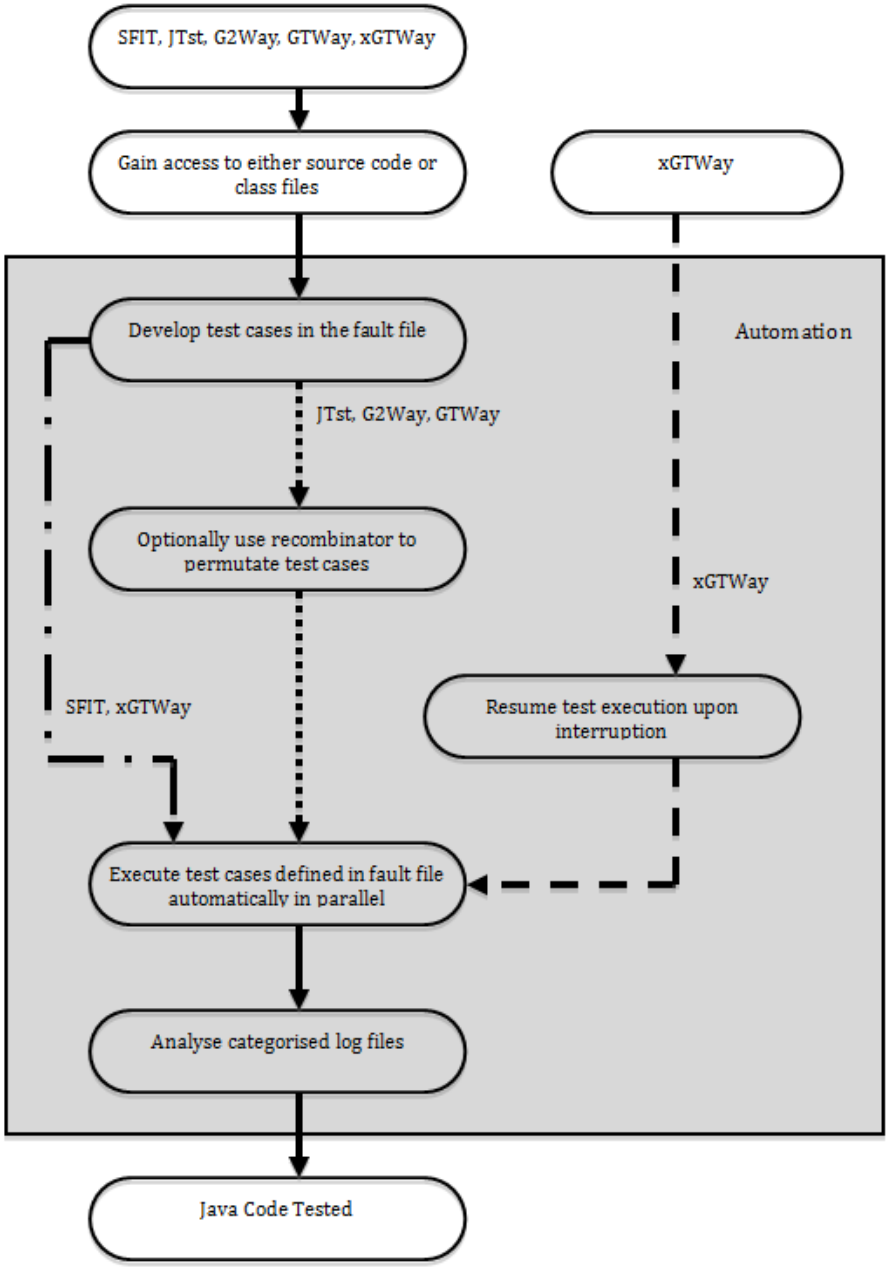

Fig. 2. Comparative Process Flow

4. Concluding Remark

In this paper, the evolution of xGTWay families of implementation has been highlighted. Although with different features, the objective of easing the burden from the testers in the testing the developed source code remains intact in each implementation. Currently, there is no single version of the family that is robust enough which have all the essential capabilities required. Further enhancement on the latest version of xGTWay can include adding the test generation feature and verifying the parallel test execution implementation in order to enhance its applicability amongst the rest of the family members.

References

[1] Hakonen H, Hyrynsalmi S, Järvi A. Reducing the Number of Unit Tests with Design by Contract. in *Proceedings of the 12th International Conference on Computer Systems and Technologies* 2011; 161-166.

[2] Brown MK. A Framework for Parallel Unit Testings: Work in Progress in *Proceedings of the 48th Annual Southeast Regional Conference* 2010; 1-4.

[3] Bertolino A. Software Testing Research: Achievements, Challenges, Dreams in *Future of Software Engineering Conference* 2007; 85-103.

[4] Berner S, Weber R, Keller RK. Observations and Lessons Learned From Automated Testing in *Proceedings of the 27th International Conference on Software Engineering* 2005; 571-579.

[5] Stobie K. Too Darned Big to Test *Queue* 2005; 3: 30-37.

[6] Alang Hassan MD, Zamli KZ, Isa NAM. SFIT – A Software Fault Injection Tool in *Proceedings of The First Malaysian Software Engineering Conference (MySEC'05)* 2005; 260-262.

[7] Zamli KZ, Isa NAM. JTst – An Automated Unit Testing Tool for Java Program *American Journal of Applied Sciences* 2008; 5: 77-82.

[8] Klaib MFJ, Zamli KZ, Isa NAM, Younis MI, Abdullah R. G2Way A Backtracking Strategy for Pairwise Test Data Generation in *Proceedings of 15th Asia-Pacific Software Engineering Conference* 2008; 463-470.

[9] Zamli KZ, Klaib MFJ, Younis MI, Isa NAM, Abdullah R. Design And Implementation Of A T-Way Test Data Generation Strategy With Automated Execution Tool Support *Information Sciences* 2011; 181: 1741-1758

[10] Alang Hassan MD. Enhancing and Evaulating a Software Fault Injection Tool (SFIT) *Master Thesis* 2005; Universiti Sains Malaysia.

[11] Zamli KZ, Alang Hassan MD, Isa NAM, Fadel JKM. Development of an Automated Testing Tool for Java® Programme *ASM Science Journal* 2007; 1: 87-100.

[12] Zamli KZ, Hassan MDA, Isa NAM, Azizan SN. An Automated Software Fault Injection Tool for Robustness Assessment of Java COTs in *Proceedings of International Conference on Computing &Informatics (ICOCI '06)* 2006; 1-6.

[13] Zamli KZ, Isa NAM, Klaib MFJ, Azizan SN. A Tool for Automated Test Data Generation (and Execution) Based on Combinatorial Approach *International Journal of Software Engineering and Its Applications* 2007; 1: 19-35.

[14] Zamli KZ, Isa NAM, Klaib MFJ, Azizan SN. Designing A Combinatorial Java Unit Testing Tool in *Proceedings of the Third Conference on IASTED International Conference: Advances in Computer Science and Technology (ACST'07)* 2007; 153-158.

[15] Klaib MFJ. Development Of An Automated Test Data Generation And Execution Strategy Using Combinatorial Approach *PhD Thesis* 2009; Universiti Sains Malaysia.

[16] Younis MI, Zamli KZ, Isa NAM. MIPOG - Modification Of The IPOG Strategy For T-Way Software Testing in *Proceeding of The Distributed Frameworks and Applications (DFmA)* 2008.

[17] Younis MI. MIPOG: A Parallel T-Way Minimization Strategy For Combinatorial Testing *PhD Thesis*2010; Universiti Sains Malaysia.

[18] Younis MI, Zamli KZ. MC-MIPOG: A Parallel T-Way Test Generation Strategy For Multicore Systems *ETRI Journal* 2010; 32: 73-83.

[19] Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J. IPOG: A General Strategy For T-Way Software Testing in *Proceedings of the 14th Annual IEEE International Conference and Workshops on The Engineering of Computer-Based Systems* 2007; 549-556.

[20] Lei Y, Kacker R, Kuhn R, Okun V, Lawrence J. IPOG/IPOG-D: Efficient Test Generation For Multi-Way Combinatorial Testing *Journal of Software Testing, Verification and Reliability* 2008; 18: 125-148.