CHAPTER 1

INTRODUCTION

Computing technology has gone a long way since the first Babbage computer. Today, many chores that were once manual have been taken over by computersoftware. Our dependencies on software raise fundamental issues on quality and reliability. Here, software testing becomes immensely important. Providing confidence, identifying weaknesses, imposing an acceptable degree of quality as well as establishing the extent to which the requirements have been met are amongst the reasons for software testing (Alsewari,et al., 2012).

"Software testing is the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item It is a standard, though imperfect, method of assuring software quality ".

Of the primary purposes of testing are to detection of software failures so that defects may be discovered and corrected, quality assurance, verification and validation and reliability estimation (Pan, 1999).

Testing cannot establish that product functions properly under all conditions but can only establish that it does not function properly under specific conditions (Kaner, et al, 1999).

The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do?. Software testing is a trade-off between budget, time and quality(Software testing,[online]Availablefrom:http://en.wikipedia.org /wiki/ Software testing).

Testing is expensive part of software development. It often consumes between 1/3 and 1/2 of the total cost of software development. Although it would be ideal to use as many test cases as possible, this is impractical since the total number of possible test cases is usually prohibitively large. Therefore, new approaches are required to generate test sets that are substantially smaller than exhaustive test sets but highly effective at detecting faults (Kobayashi, et al, 2001).

Exhaustive testing of computer software is intractable, it’s completely infeasible even for a small program, with a relatively simple set of variables and relatively few possible states per variable and the total number of possible valid states in combination is intractably large. To arrive at that number, we consider each variable, and count the number of valid states for it. We then multiply the numbers of all those valid states for each variable together. But empirical studies of software failures suggest that testing can in some cases be effectively exhaustive. Studies show that software failures in a variety of domains were caused by combinations of relatively few conditions. These results have important implications for testing. If all faults in a system can be triggered by a combination of n or fewer parameters, then testing all n-tuples of parameters is effectively equivalent to exhaustive testing, if software behavior is not dependent on complex event sequences and variables have a small set of discrete values (Kuhn, 2004).

In testing, we want to be sure that we don't miss problems based on conflicts between two or more conditions, variables, or configurations, so we often test in combinations in order to find defects most efficiently. In formal mathematics, the study of combinations is called "*combinatorics*", combinations are, formally, "selections of a number of different items from a set of distinguishable items when the order of selection is ignored”. (Pairwise Testing,[online] Available from:http://www.developsense.com/pairwiseTesting.html, [accessed November 2007]).

Combinatorial testing, which has proven very effective in fault detection, is a testing strategy that applies the theory of combinatorial design to test software systems. Given a system under test with k parameters, t-way combinatorial testing requires all combinations of values of t (out of k) parameters be covered at least once, where t is usually a small integer. If test parameters are modeled properly, all faults caused by interactions involving no more than t parameters will be detected (N.Borazjany, et al, 2012).

To apply combinatorial testing, it is necessary to find a set of test inputs that covers all t-way combinations of parameter values, and to match up each set of inputs with the expected output for these input values. These are both difficult problems, but they can now be solved with new algorithms on currently available hardware (Automated Combinatorial Testing for Software (ACTS).[Online] Available from:http://www.nist.gov/itl/csd/scm/acts.cfm).

Pairwise testing is a widely popular approach to combinatorial testing problems.The number of articles and text books covering the topic continues to grow as do the number of commercial and academic courses that teach the technique (Bach, J. Schroeder, 2004).

What has made combinatorial testing practical today is the development of efficient algorithms to generate tests covering t-way combinations, and effective methods of integrating the tests produced into the testing process. A variety of approaches can be used to make combinatorial testing practical and effective addition to the software tester's toolbox.

There are basically two approaches to combinatorial testing that use combinations of configuration parameter values, or combinations of input parameter values. In the first achieving combinatorial coverage of configuration parameter values, in the second approach, we select combinations of input data values, which then become part of complete test cases, creating a test suite for the application (Practical Combinatorial Testing.[Online] Available from:csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf[accessedOctober 2010]).

Combinatorial test data generators generate data tables for testing. The most basic, commonly used combinatorial data generation strategy is what is known as pairwise testing, all-pairs testing, covering arrays.

The theoretical basis for pairwise testing is what is known as coupling effect. This is a practical hypothesis that software faults can be discovered by relatively simple tests. How this is related to pairwise testing is that the coupling effect hypothesis suggests that if there is a fault that manifests with a specific setting of configuration variables, it is most likely caused actually by only a small subset of those variable values.

Of course, there is no reason why coupling two variables and no more would be always the best strategy. A natural extension of pairwise testing is indeed to cover not only pairs but also triples, quartets and so on. This is not necessarily good to do for all small subsets of data variables, so advanced combinatorial data generation tools allow users to define the “strength” of data combination individually for different data variables and their combinations(Huima, 2012).

To cover all pair-wise combinations of parameter values, we need only a small number of test cases if we select them appropriately. Thus the use of this approach can lead to reduced cost of testing

Pairwise testing has become an indispensable tool in a software tester’s toolbox. The technique has been known for almost 20 years, but it is only in the last few years that we have seen a tremendous increase in its popularity(Czerwonka, 2008).

Combinatorial data generation is a very good way to generate discrete test data tables and combinatorial testing is a method that can reduce cost and increase the effectiveness of software testing for many applications.

1.2Problem Statement

Combinatorial testing is a problem that show whenever we have a product that processes multiple variables that may interact .The variables may come from a variety of sources ,such as user interface ,operating system , peripherals , database or from across a network .

The task in combinatorial testing goes beyond testing individual variables and verify that different combinations of variable are handled correctly by the system.

Compared to extensive work that has been reported on the theoretical side, there is a lack of empirical studies and experience reports on applying combinatorial testing to real- life systems. This research pays special attention to usability of the pairwise-testing technique. It refer to specific case study, it does not describe any radically new method of efficient generation of pairwise test suites, a topic that has already been researched extensively. Pairwise testing approach must be modified to become practically applicable.

1.3 Objective of the research

The purpose is to know how to apply combinatorial testing in practice and to evaluate the effectiveness of combinatorial testing applied to a real-life system.

The Objective of this research is:

1.3.1 Provide a practical way to detect failures caused by parameter interactions with trade –off between cost and efficiency.

1.3.2 Selects test cases with sampling mechanism to systematically cover parameter value combinations using a small test set which is relatively easy to manage and execute.

1.4 Research scope

The scope of this research is in the area of combinatorial testing. The idea is to apply combinatorial testing in practice by using combinatorial test generation tool called PictMaster (http://en.sourceforge.jp/projects/pictmaster, 2013).Then present a case study of applying combinatorial testing. Automation testing increases the test coverage, improve accuracy, saves time and money in comparison to manual testing. There are a number of automatic test case generation tools available, but these can suffer from combinatorial explosions in the number of possibilities to test.

1.5 Research methodology and tools

PictMaster is a tool that generates a test case for several types of tests, including a combination test. Its Excel-based free software that improved PICT (Pairwise Independent Combinatorial Testing Tool) making it easier and more sophisticated. PictMaster is free generating combination test cases that use the Pairwise method.

PICT itself is an application based on CUI (Character User Interface) to run on the command prompt. Now, most people are unfamiliar with the command prompt. There may be a lot of users who do not want to use PICT working in command prompt, generating a combination test case in Excel would be very useful. The Excel workbook, PictMaster, realized this useful mechanism. PictMaster overlays the CUI-based PICT with an Excel GUI (Graphical User Interface) based shell. This flexibility of the generation algorithm allows for adding interesting new features easily, the algorithm is also quite effective, it is fast enough for all practical purposes.

The steps are identify the test parameters based on system characteristics, test values and System under Test (SUT). Generate test cases and control them by use

Sub model, extend sub model, constrains, proto type .then get expected result(http://en.sourceforge.jp/projects/pictmaster, 2013).

1.6Research organization

The remainder of this Research is organized as follows.

- **Chapter 2** illustrates the literature review.
- **Chapter 3** implement combinatorial methods in model uses pairwise testing, (Case Study ElectronicRegistration program at the University of IslamicOmdurman).
- **Chapter 4**shows result and discussion.
- **Chapter 5** Conclusion and future work.

CHAPTER 2

LITERATURE REVIEW

A combinatorial search problem is one where an initial state is to be transformed into a goal state by application of a series of operators, such as assignment of values to variables. The space of possible states is usually exponential in the size of the input and finding a solution is NP-problem. A common way of solving such problems is to use heuristics. A heuristic is a strategy that determines which operators to apply when. Heuristics are not necessarily complete or deterministic, they are not guaranteed to find a solution if it exists or to always make the same decision under the same circumstances. The nature of heuristics makes them particularly amenable to Algorithm Selection.Choosing a heuristic manually is difficult even for experts, but choosing the correct one can improve performance significantly.

An algorithm may be a system, a programme, a heuristic, a classifier or a configuration. This is not made explicit unless it is relevant in the particular context (Kotthoff, 2012).

Combinatorial testing refers to a testing strategy that applies the principles of combinatorial design to the domain software test generation. It creates tests by combining parameter values with combinatorial test generation strategies.

Cohen et al. proposed a strategy called Automatic Efficient Test Generator (or AETG), which constructs a test set by repeatedly adding one test at a time until all the combinations of parameter values are covered. A greedy algorithm is used to construct the tests such that each test covers as many uncovered combinations as possible. Several variants of this strategy have been reported in the literature. These variants share the same framework as AETG but use different heuristics for the greedy construction of each test.

Leietal(Lei, Tai, 1998) (Lei, Tai, 2002), proposed the IPO (In-Parameter-Order) strategy, which builds a pairwise test set for the first two parameters, extends the test set to cover the first three parameters, and continues to extend the test set until it builds a pairwise test set for all the parameters (Lei et al ., 2007).

Higher interaction strength in the development of IPOG(In-Parameter-Order-General). Jenny generates test data in a number of stages. Firstly, Jenny generates test data to cover all the 1-way interaction. Then, Jenny will extend the first stage test data to greedily cover the 2-way interactions. Optionally, this process can continue until the nth-way interactions as specified by the user. Covering one parameter at a time allows the IPO strategy to achieve a lower order of complexity than AETG(Alsewari,et al., 2012).

Most recently, heuristic search techniques such as hill climbing and simulated annealing have been applied to multi-way testing. Unlike AETG and IPO, which builds a test set from scratch, heuristic search techniques start from a pre-existing test set and then apply a series of transformations to the test set until a test set is reached that covers all the combinations. Heuristic search techniques can produce smaller test sets than AETG and IPO, but they typically take longer to complete (Lei et al., 2007).

Pair-wise testing is an important testing approach. This is a type of combinatorial testing which requires that for each pair of input parameters of a system, every combination of valid values of these parameters be covered by at least one test case. Studies have shown pair-wise testing to be a very practical and effective software testing criterion (Alton, et al.,2012).

In general, existing interaction strategies for pairwise testing can be categorized into two categories based on the dominant approaches, that is, algebraic approaches or computational approaches. Algebraic approaches construct test sets using pre-defined rules or mathematical function. Thus, the computations involved in algebraic approaches are typically lightweight, and in some cases, algebraic approaches can produce the most optimal test sets. However, the applicability of algebraic approaches is often restricted to small configurations. Orthogonal arrays (OA), use mathematics of arrays (MOA) and TConfig are typical example of the strategies that are based on algebraic approach. Unlike algebraic approaches, computational approaches often rely on the generation of the all pair combinations. Based on all pair combinations, the computational approaches iteratively search the combinations space to generate the required test case until all pairs have been covered. In this manner, computational approaches can ideally be applicable even in large system configurations. However, in the case where the number of pairs to be considered is significantly large, adopting computational approaches can be expensive due to the need to consider explicit enumeration from all the combination space.

For PICT (Pairwise Independent Combinatorial Testing tool), it first generates all the specified interaction before and randomly selecting their corresponding interaction combinations to form the test cases as part of the complete test suite. All pair's strategy, TVG and CTE_XL share the same property as far as producing deterministic test cases is concerned although little is known about the actual algorithms employed due to limited availability of references. A more recent strategies based on computational approaches are IRPS, and G2Way. IRPS is deterministic in nature and focuses on efficient data structure for storing and searching pairs. In this manner, IRPS gives relatively fast execution time as compared to other strategies. G2Way adopts a backtracking algorithm to merge combinable pairs in order to generate the pairwise test suite.In a nut shell, SA adopts a probability-based transformation equation along with a greedy binary search algorithm to iteratively find the best test case to cover all the required (pairwise) interactions from a random search space. In similar manner, PPSTG, a PSO based strategy, iteratively performs local and global searches to find the candidate solution to be added to the final suite until all the pairwise interactions are covered. Table (2-1), (2-2)(Alsewari,et al., 2012).

Table (2-1) :show test suite size for configuration with 10 V-valued parameters

V	TVG	PICT	CTE-XL	Tconfig	IPOG	Jenny	PPSTG	PHSS
3	18	18	18	17*	20	19	17*	17*
4	33	31	33	31	31	30	29	28*
5	50	47	50	48	50	45	45	43*
6	72	66	71	64	68	62	62	60*
7	98	88	97	85	90	83	81	79*
8	124	112	125	114	117	104*	109	105
9	152	139	161	139	142	129	139	127*
10	189	170	192	170	176	157	170	155*

Table (2-2) :show test suite size for a configuration with p 2-value parameters

P	TVG	PICT	CTE-XL	Tconfig	IPOG	Jenny	PPSTG	PHSS
3	4*	4*	6	4*	4*	5	4*	4*
4	6	5*	6	6	6	6	6	6
5	6*	7	6*	6*	6*	7	6*	6*
6	6*	6*	8	7	8	8	7	7
7	8	7*	8	9	8	8	7*	7*
8	8	7*	8	9	8	8	8	8
9	8*	9	9	9	8*	8*	8*	8*
10	9	9	9	9	10	10	8*	8*
11	9	9	10	9	10	9	9	8*
12	10	9*	10	9*	10	10	9*	9*
13	10	9*	10	9*	10	10	9*	9*
14	10	10	10	9*	10	10	9*	10
15	10	10	10	9*	10	10	10	10

*Cells with asterisk (*) in table (2-1)-(2-2) show the smallest generated size of the test suite by each strategy.

CHAPTER 3

Practical Combinatorial Testing

The demand for multi-functional software has grown drastically over the years. To cater for this demand, software engineers are forced to develop complex software with increasing number of input parameters. As a result, more and more dependencies between input parameters are to be expected, openingmore possibilities of faults due to interactions. Although traditional static and dynamic testing strategies are useful in fault detection and prevention, however they are not sufficiently effective to detect faults due to interaction. As a result, many researchers nowadays are focusing on sampling strategy that is based on interaction testing (termed t-way testing strategies where t indicates the interaction strength).

The electronic registration system is one of the most important systems where the IslamicOmdurman Universityoffers several services to students, colleges, and financial management and converts operations relating to the registration of the student to the digital environment.

It is important to note that the programs in the electronic registration are relatively small, in terms of lines of code, and have a small number of input parameters, Its abstract models contains 10 abstract parameters and 8 constraints.

3.1 APPROACH

This section, explainsthe approach to apply combinatorial testing. The approach consists of threeprocesses:

3.1.1 Create an abstract model.

3.1.2 Generate an abstract test set.

3.1.3 Get Expected results.

Create abstract model this step has two major tasks:

- Define abstract parameters and values.
- Define relations and constraints.

3.1.1.1 Define abstract parameters and values:

First, Ianalysed the system specification and identify factors that may affect the behaviour of the system. These factors are candidates for abstract parameters.Consider a running example of an electronic registration Application as shown inFigure(3-1), (3-2).there are many parameters for the user to insert or choose in orderto complete the registration process for the studentin this application.

For simplification I use symbolic values Table (3-1).Conveniently, as seen in Table (3-2), the electronic registration option representation can also be translated into a table of 10 columns (or parameters) and 2,3,4,5 rows (or values).

Figure (3-1): Electronic Registration Application (1)

Figure (3-2): Electronic Registration Application (2)

Table (3-1): Parameters and Values Conversion

ActualParameters and Their Values	Symbolic Representations
University ID {correct number, wrong number }	No = {v1,v2}
Name {correct name, wrong name}	Na = {n1, n2}
Type Acceptance{ general, private, external ,mature study, Darfur student }	Acc = { a1,a2,a3,a4,a5}
Academic position {success, freeze, role of the second, repeat }	Acad ={ s1,s2,s3,s4}
Registration fee {specific, unspecified }	Reg ={ p1,p2}
Tuition fees { full fees, half fees , no fees }	Fee = { f1,f2,f3}
Exception { exempt , non- exempt }	Exe = { y1,y2}
Payment Type { first premium, last premium, other fees}	Pay = { k1,k2,k3}
Type of college { scientific , theoretical }	Coll = { l1,l2}
The academic year {correct year ,wrong year }	Year = { b1,b2}

Table (3-2): Base Data Values

	Input Variables									
	No	**Na**	**Acc**	**Acad**	**Reg**	**Fee**	**Exe**	**Pay**	**Coll**	**Year**
BASE VALUES	v1	n1	a1	s1	p1	f1	y1	k1	l1	b1
	v2	n2	a2	s2	p2	f2	y2	k2	l2	b2
			a3	s3		f3		k3		
			a4	s4						
			a5							

3.1.1.2 Define relations and constraints

Relations are used to create parameter groups that can be covered at different strengths. Furthermore, parameters in different groups are independent and thus their combinations do not have to be tested.I use the default relation where all the parameters are considered to be in the same group. The parameters for input data could be put into one group. Constraints are used to exclude combinations that are not valid from the domain semantics. A total of7 constraints are specifiedTable (3-3). All these 7 constraints are concerned with the position values of different parameters.

Table (3-3): Application Constrains

No.	Constraints
1	In the case of External Acceptance the student should pay the enrolling + study fee in the first instalment.
2	If the student has passed and the Acceptance (general– private – mature study) at theoretical faculty, the student should pay enrolling +50% of study fee in the first instalment.
3	If the student has passed and the Acceptance (general – private – mature study) at scientific faculty, the student should pay enrolling +50% of study fee in another instalment.
4	If the Academics Situation of student is repeat, the student should pay the enrolling fee + 50% of study fee in the last instalment.
5	If the student is Accepted from Darfur Student should be exempted from the fee.
6	In the case of Academic freeze the student should only pay the enrolling fee in the last instalment.
7	The name, university number and year of study should be inserted correctly.

3.1.2.1 Generate an abstract test set

Generate abstract tests in this step, an abstract test set is generated using an existing combinatorial test generation tool. I used the PictMaster tool. PictMaster can generate a combinatorial test set with strength 2 through 6. Firstbegan introducingparameters and values in the tool to build my model Figure(3-3).

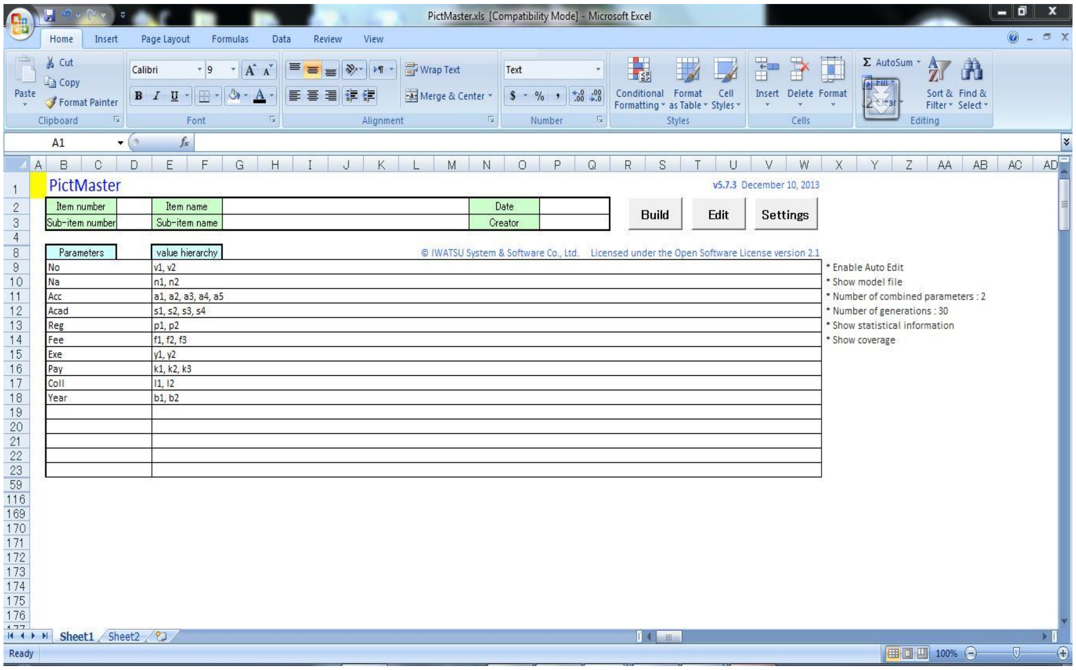


Figure (3-3): parameters and value hierarchy columns

Then selected settings Figure (3-4)

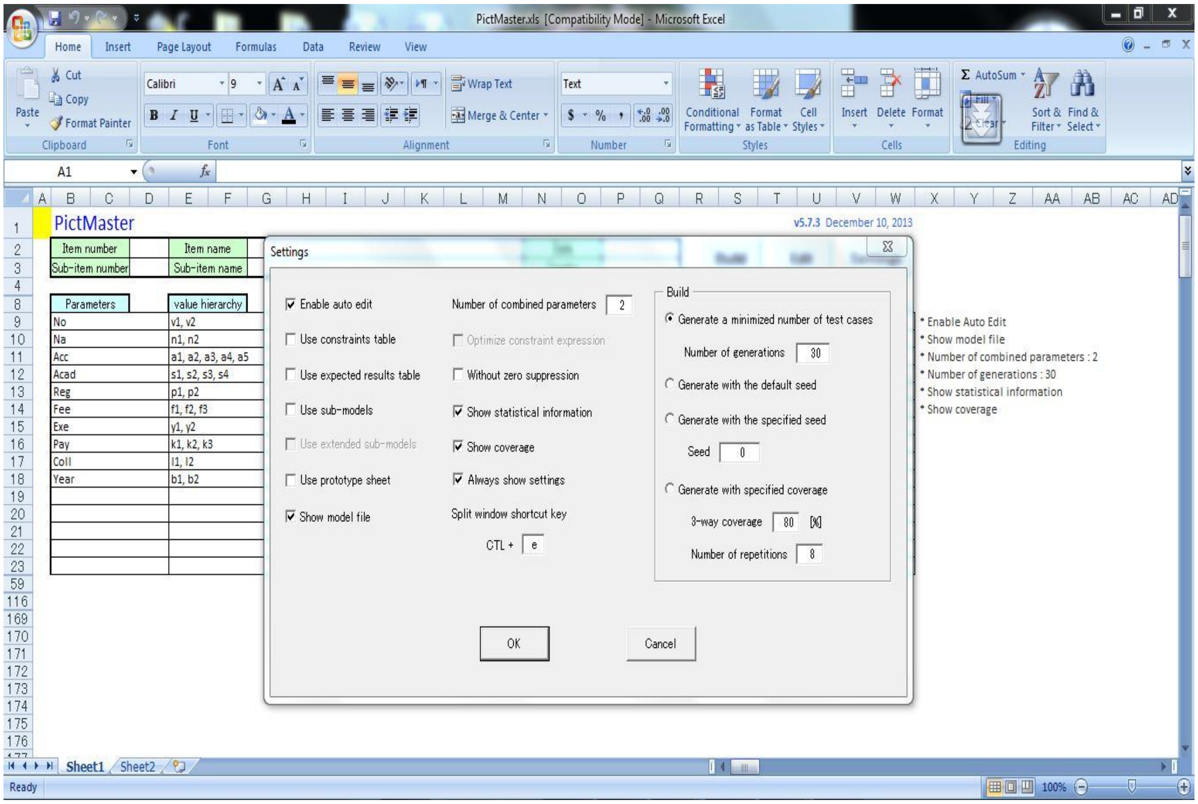
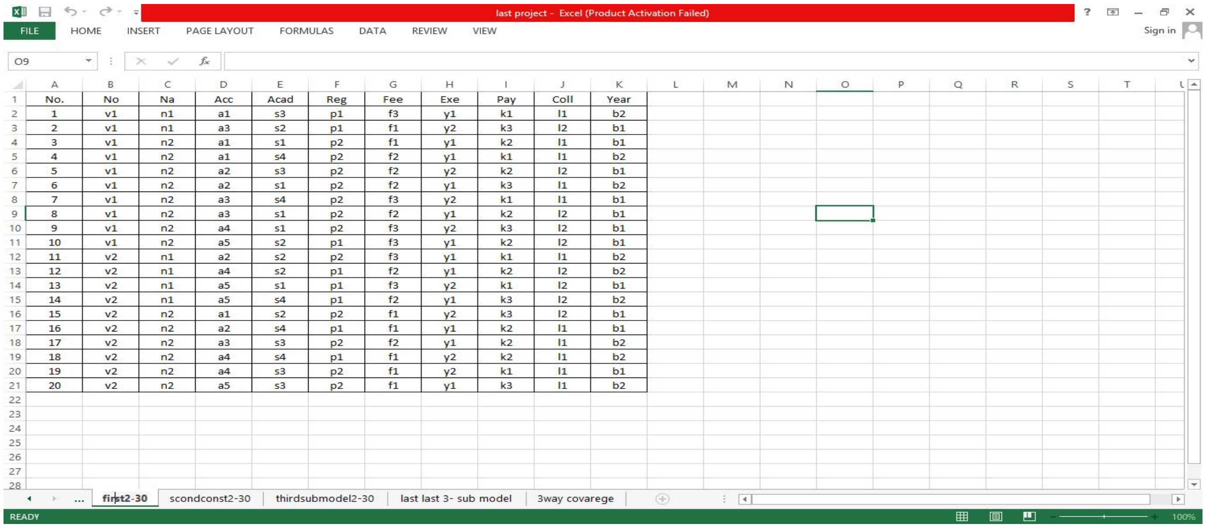


Figure (3-4): setting window

And pressed build button to get the test cases Figure (3-5):



No.	No	Na	Acc	Acad	Reg	Fee	Exe	Pay	Coll	Year
1	v1	n1	a1	s3	p1	f3	y1	k1	l1	b2
2	v1	n1	a3	s2	p1	f1	y2	k3	l2	b1
3	v1	n2	a1	s1	p2	f1	y1	k2	l1	b1
4	v1	n2	a1	s4	p2	f2	y1	k1	l1	b2
5	v1	n2	a2	s3	p2	f2	y2	k2	l2	b1
6	v1	n2	a2	s1	p2	f2	y1	k3	l1	b2
7	v1	n2	a3	s4	p2	f3	y2	k1	l1	b1
8	v1	n2	a3	s1	p2	f2	y1	k2	l2	b1
9	v1	n2	a4	s1	p2	f3	y2	k3	l2	b1
10	v1	n2	a5	s2	p1	f3	y1	k2	l2	b1
11	v2	n1	a2	s2	p2	f3	y1	k1	l1	b2
12	v2	n1	a4	s2	p1	f2	y1	k2	l2	b2
13	v2	n1	a5	s1	p1	f3	y2	k1	l2	b1
14	v2	n1	a5	s4	p1	f2	y1	k3	l2	b2
15	v2	n2	a1	s2	p2	f1	y2	k3	l2	b1
16	v2	n2	a2	s4	p1	f1	y1	k2	l1	b1
17	v2	n2	a3	s3	p2	f2	y1	k2	l1	b2
18	v2	n2	a4	s4	p1	f1	y2	k2	l1	b2
19	v2	n2	a4	s3	p2	f1	y2	k1	l1	b1
20	v2	n2	a5	s3	p2	f1	y1	k3	l1	b2

Figure (3-5): Test Cases

I displayed the statistical information for the one-time test case generation after the number of test cases has minimized completely (I increase the probability of reducing the number of test cases by increase the frequency of generation in the setting, usually 30 generations may be enough)to get the frequency of generation; the minimized, maximized, or initial number of test cases; the minimum seed value; and the elapsed time for the test cases generation Figure (3-6).

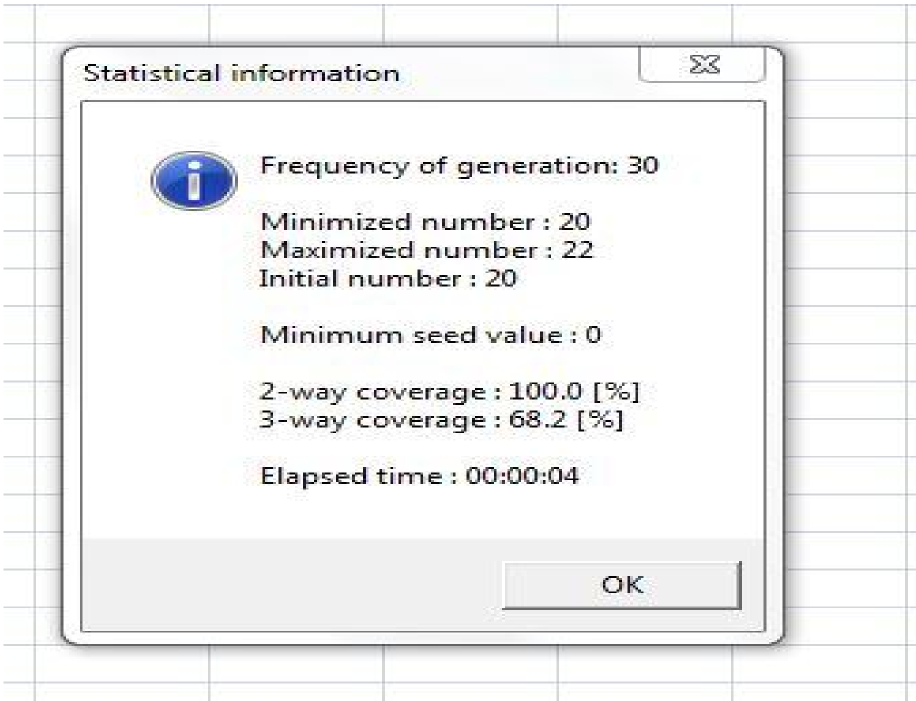


Figure (3-6): statistical information

N-way coverage (combination coverage proportion of n-parameter interactions) and t-way coverage, which have been created during the test cases generation, as Figure (3-6). The value n is defined as the specified value in the "Number of combined parameters." in the setting window the value t equals the value ofn+1. To compute the n-way coverage, the combination is generated once with n-parameter interactions. To compute the t-way coverage, the combination is generated once with t-parameter interactions.

PictMaster created the model file based on the parameters column, value hierarchy column and constraints. PictMaster transmitted the model file a.txt to PICT and the file is shown in Notepad format (first param)Figure (3-7).

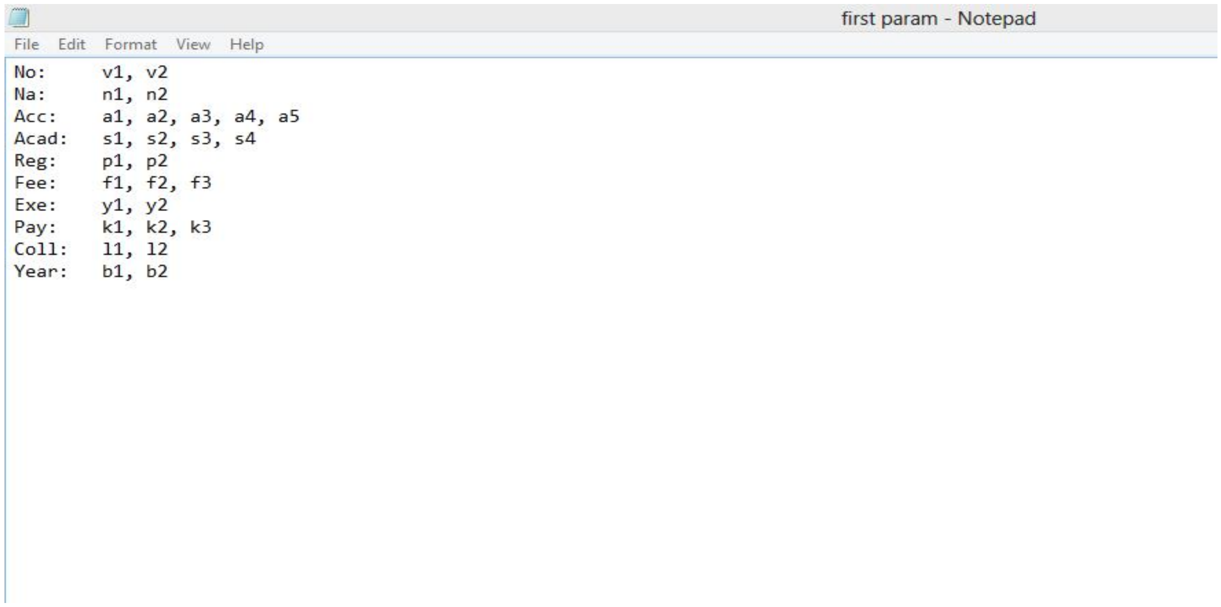


Figure (3-7): Notepad format

To improve the test cases that obtained by adding constraints, used constrains table in the setting Figure (3-8).

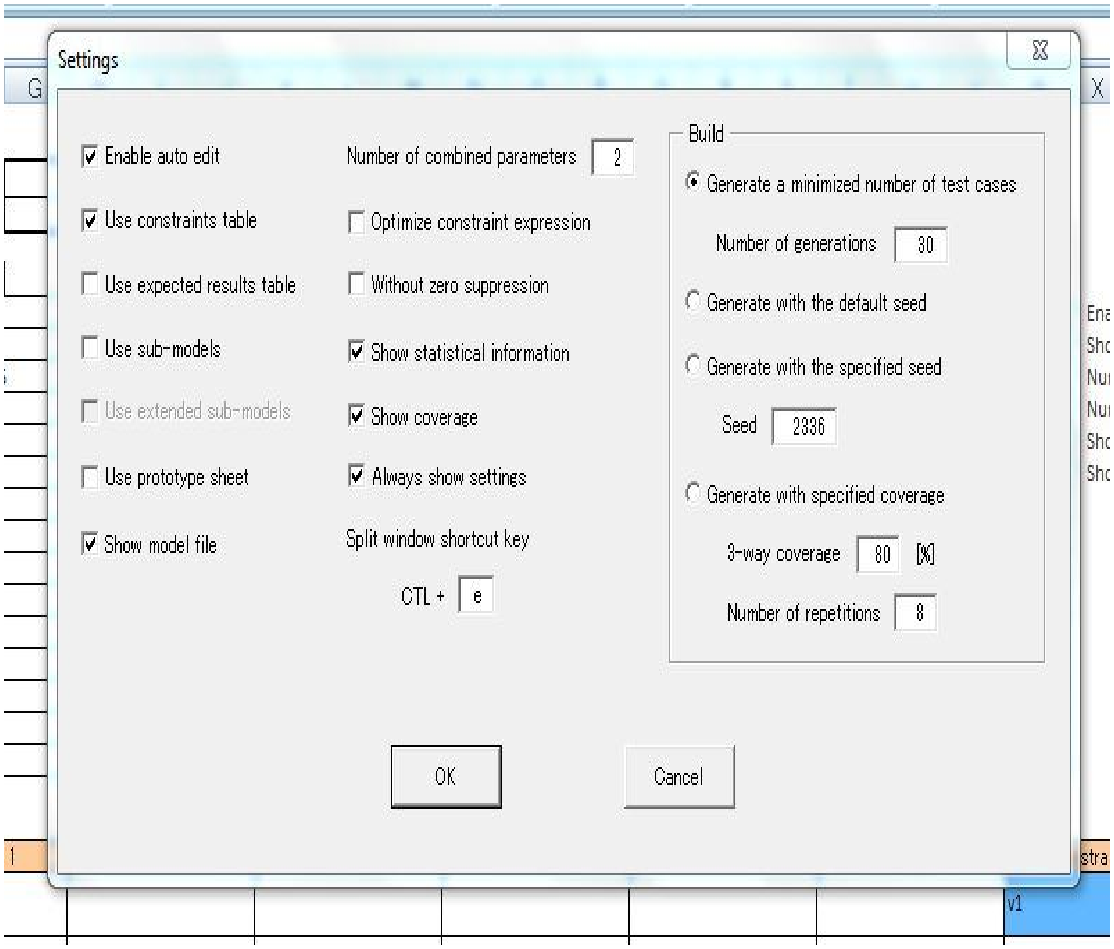
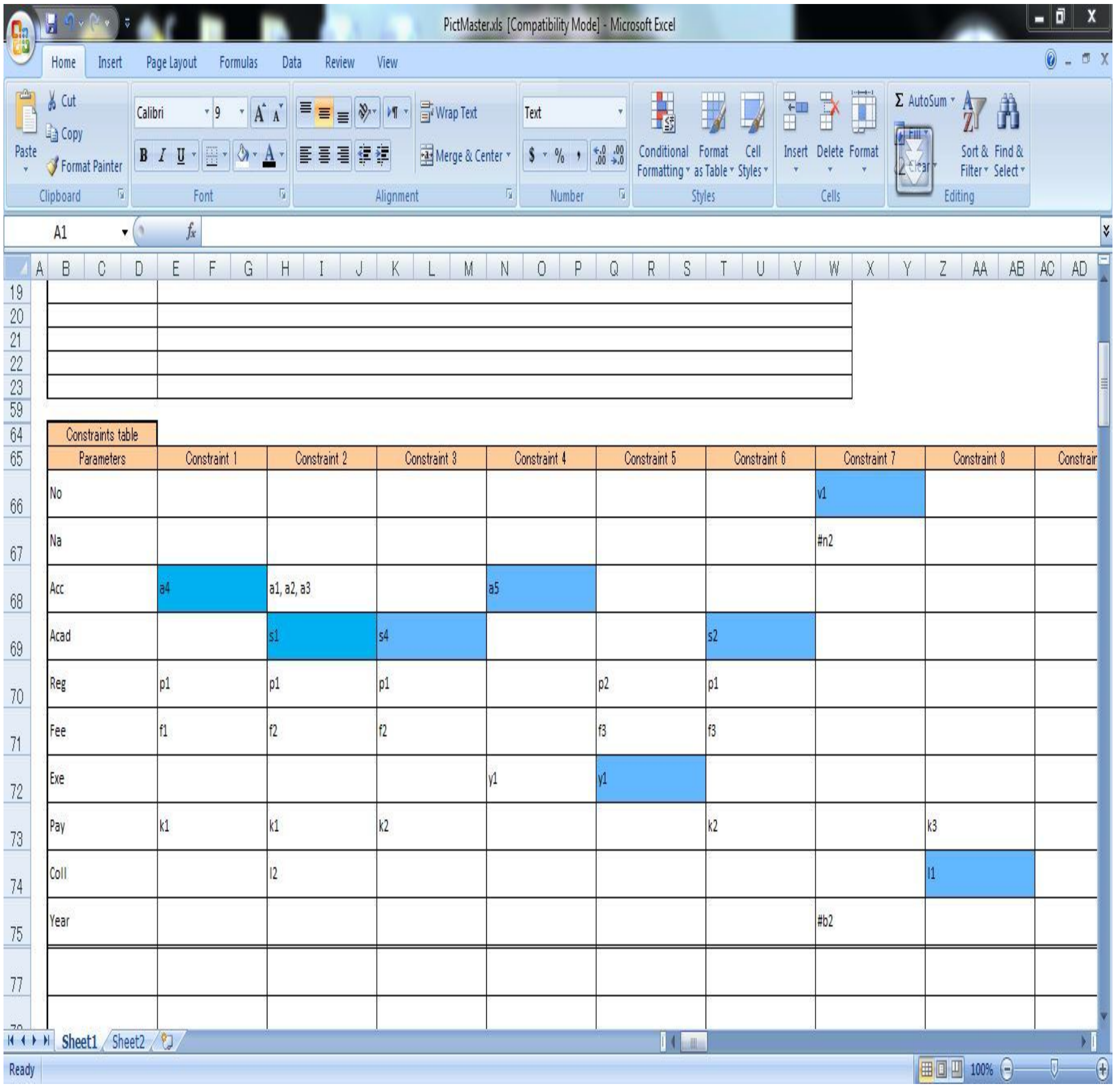


Figure (3-8): Constrains Tablesetting window

Translated the constraints in table 3 to the tool Figure (3-9).

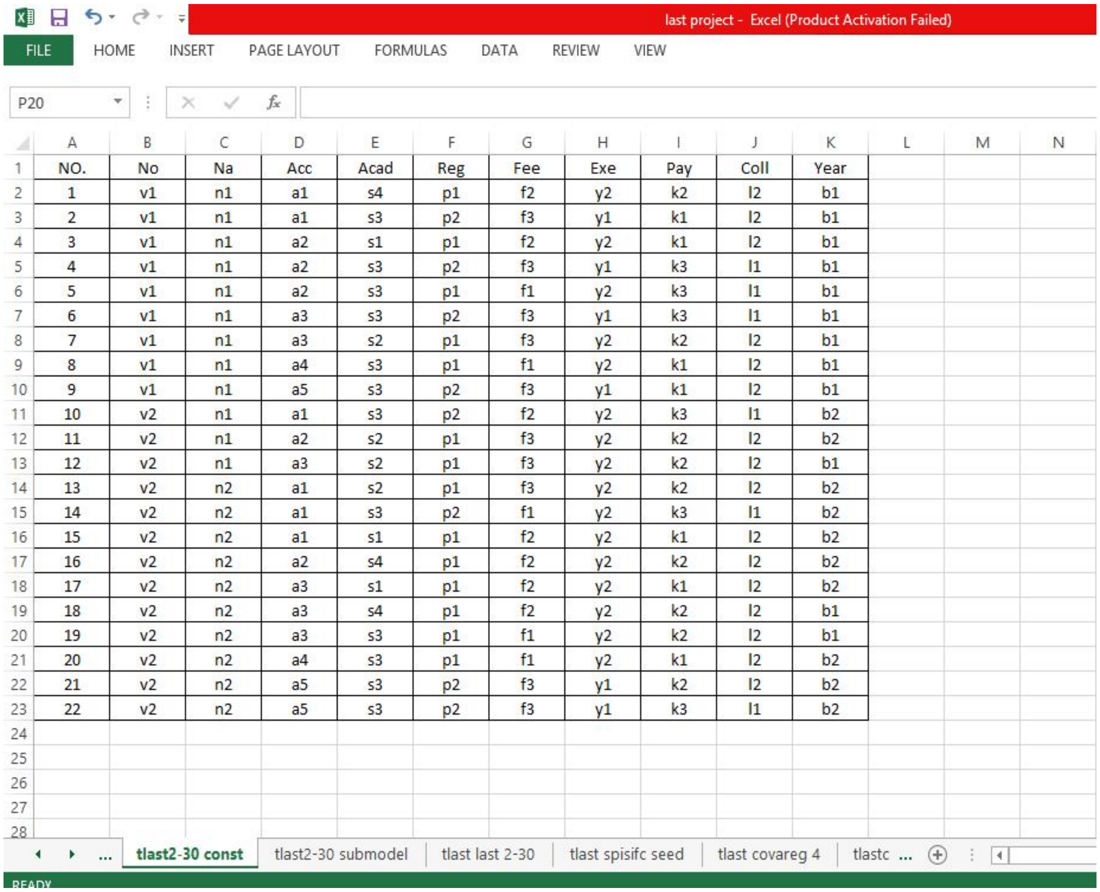


Constraints table

Parameters	Constraint 1	Constraint 2	Constraint 3	Constraint 4	Constraint 5	Constraint 6	Constraint 7	Constraint 8	Constrain
No							v1		
Na							#n2		
Acc	a4	a1, a2, a3		a5					
Acad		s1	s4			s2			
Reg	p1	p1	p1		p2	p1			
Fee	f1	f2	f2		f3	f3			
Exe				y1	y1				
Pay	k1	k1	k2			k2		k3	
Coll		l2						l1	
Year							#b2		

Figure (3-9): Adding Constrains

Then create the new test cases Figure (3-10):



NO.	No	Na	Acc	Acad	Reg	Fee	Exe	Pay	Coll	Year
1	v1	n1	a1	s4	p1	f2	y2	k2	l2	b1
2	v1	n1	a1	s3	p2	f3	y1	k1	l2	b1
3	v1	n1	a2	s1	p1	f2	y2	k1	l2	b1
4	v1	n1	a2	s3	p2	f3	y1	k3	l1	b1
5	v1	n1	a2	s3	p1	f1	y2	k3	l1	b1
6	v1	n1	a3	s3	p2	f3	y1	k3	l1	b1
7	v1	n1	a3	s2	p1	f3	y2	k2	l2	b1
8	v1	n1	a4	s3	p1	f1	y2	k1	l2	b1
9	v1	n1	a5	s3	p2	f3	y1	k1	l2	b1
10	v2	n1	a1	s3	p2	f2	y2	k3	l1	b2
11	v2	n1	a2	s2	p1	f3	y2	k2	l2	b2
12	v2	n1	a3	s2	p1	f3	y2	k2	l2	b1
13	v2	n2	a1	s2	p1	f3	y2	k2	l2	b2
14	v2	n2	a1	s3	p2	f1	y2	k3	l1	b2
15	v2	n2	a1	s1	p1	f2	y2	k1	l2	b2
16	v2	n2	a2	s4	p1	f2	y2	k2	l2	b2
17	v2	n2	a3	s1	p1	f2	y2	k1	l2	b2
18	v2	n2	a3	s4	p1	f2	y2	k2	l2	b1
19	v2	n2	a3	s3	p1	f1	y2	k2	l2	b1
20	v2	n2	a4	s3	p1	f1	y2	k1	l2	b2
21	v2	n2	a5	s3	p2	f3	y1	k2	l2	b2
22	v2	n2	a5	s3	p2	f3	y1	k3	l1	b2

Figure (3-10): new test cases

Displayedthe statistical informationFigure (3-11).

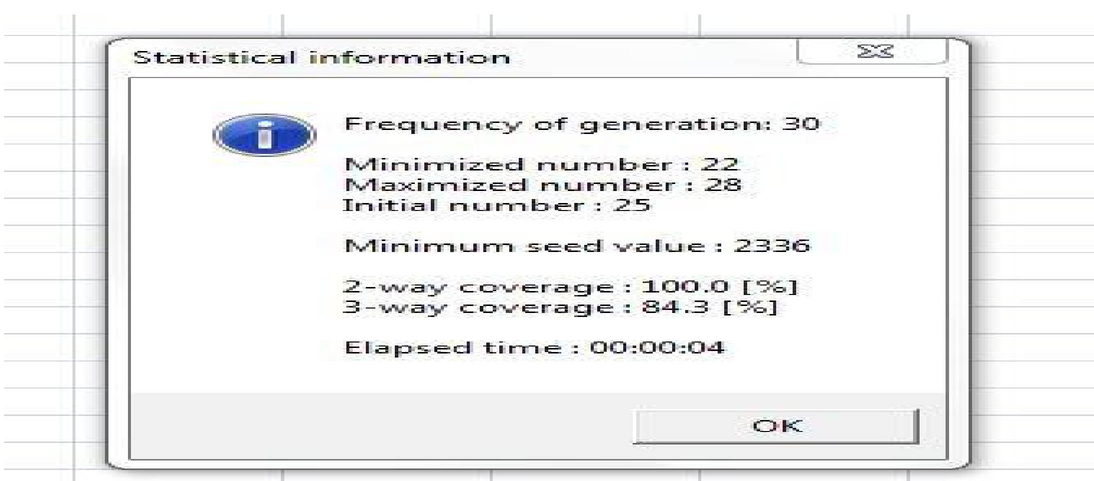


Figure (3-11): statistical information window

The model fileshown in Notepad format (scond-const)Figure (3-12).

```
No:     v1, v2
Na:     n1, n2
Acc:    a1, a2, a3, a4, a5
Acad:   s1, s2, s3, s4
Reg:    p1, p2
Fee:    f1, f2, f3
Exe:    y1, y2
Pay:    k1, k2, k3
Coll:   l1, l2
Year:   b1, b2



if ([Acc] = "a4" )
    then ([Reg] = "p1" ) and ([Fee] = "f1" ) and ([Pay] = "k1" ) ;
if ([Acad] = "s1" )
    then ([Acc] = "a1" or [Acc] = "a2" or [Acc] = "a3" ) and ([Reg] = "p1" ) and ([Fee] = "f2" ) and ([Pay] = "k1" ) and ([Coll] = "l2" ) ;
if ([Acad] = "s4" )
    then ([Reg] = "p1" ) and ([Fee] = "f2" ) and ([Pay] = "k2" ) ;
if ([Acc] = "a5" )
    then ([Exe] = "y1" ) ;
if ([Exe] = "y1" )
    then ([Reg] = "p2" ) and ([Fee] = "f3" ) ;
if ([Acad] = "s2" )
    then ([Reg] = "p1" ) and ([Fee] = "f3" ) and ([Pay] = "k2" ) ;
if ([No] = "v1" )
    then ([Na] <> "n2" ) and ([Year] <> "b2" ) ;
if ([Coll] = "l1" )
    then ([Pay] = "k3" ) ;
```

Figure (3-12): Notepad format

After that I used sub-models definition to specify certain parameters that need to be tested more, I chose (exemption and tuition fees), I used sub-models table in the settingFigure (3-13), (3-14). The multiple parameters specified in the sub-models will generate the combination of the "Number of combined parameters”.

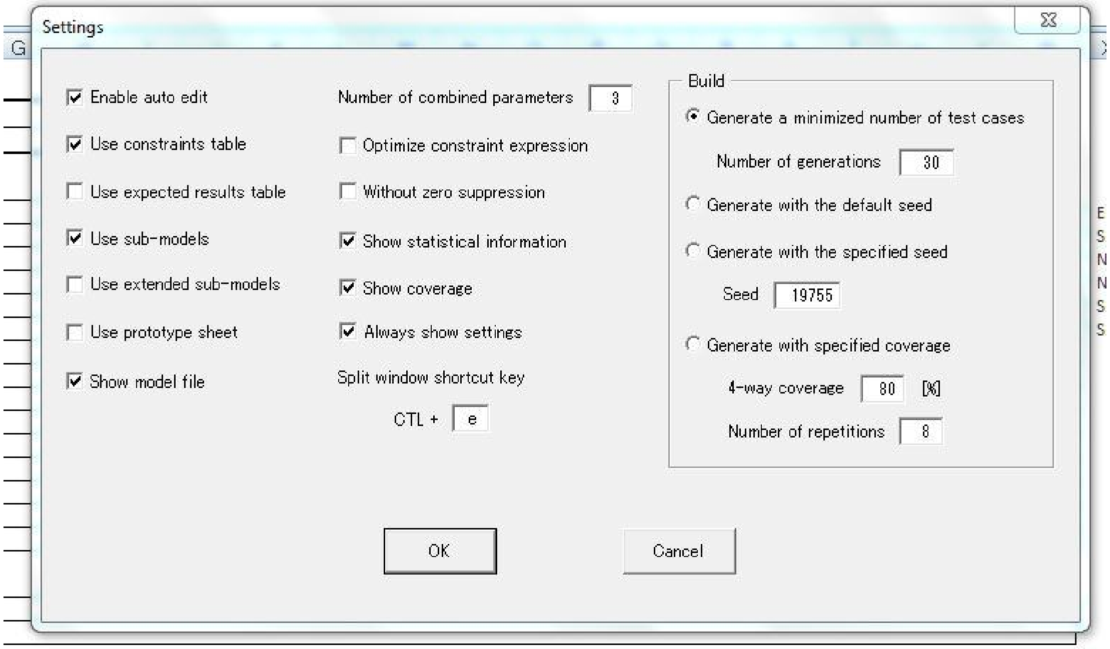


Figure (3-13): Sub-models settingwindow

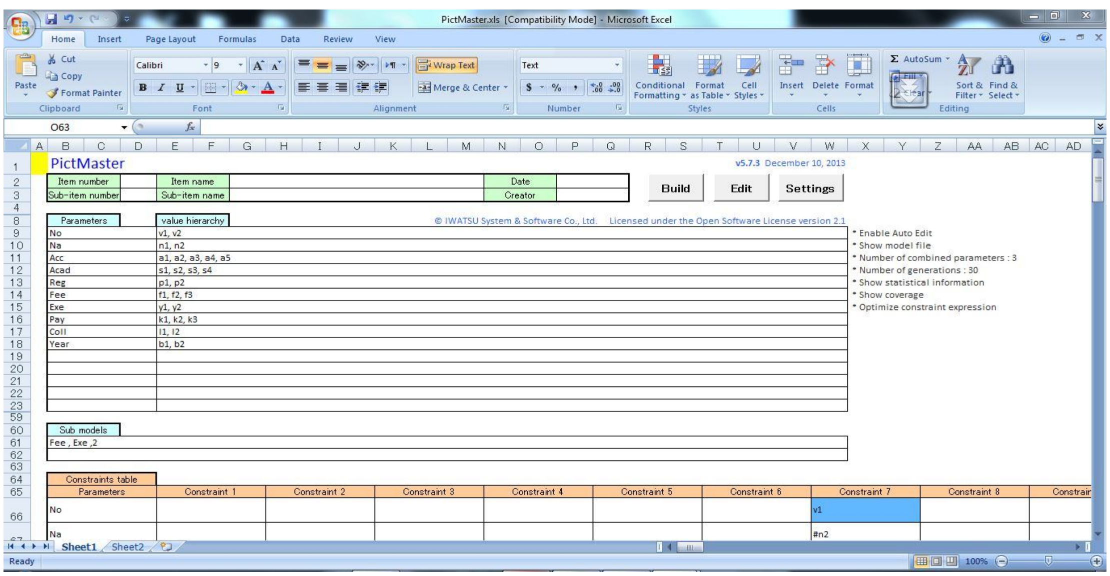


Figure (3-14): sub-models table

Then pressed build button to create the new test cases Figure (3-15),while minimizing the number of test cases, the progress bar, which is shown in Figure (3-16), is displayed.

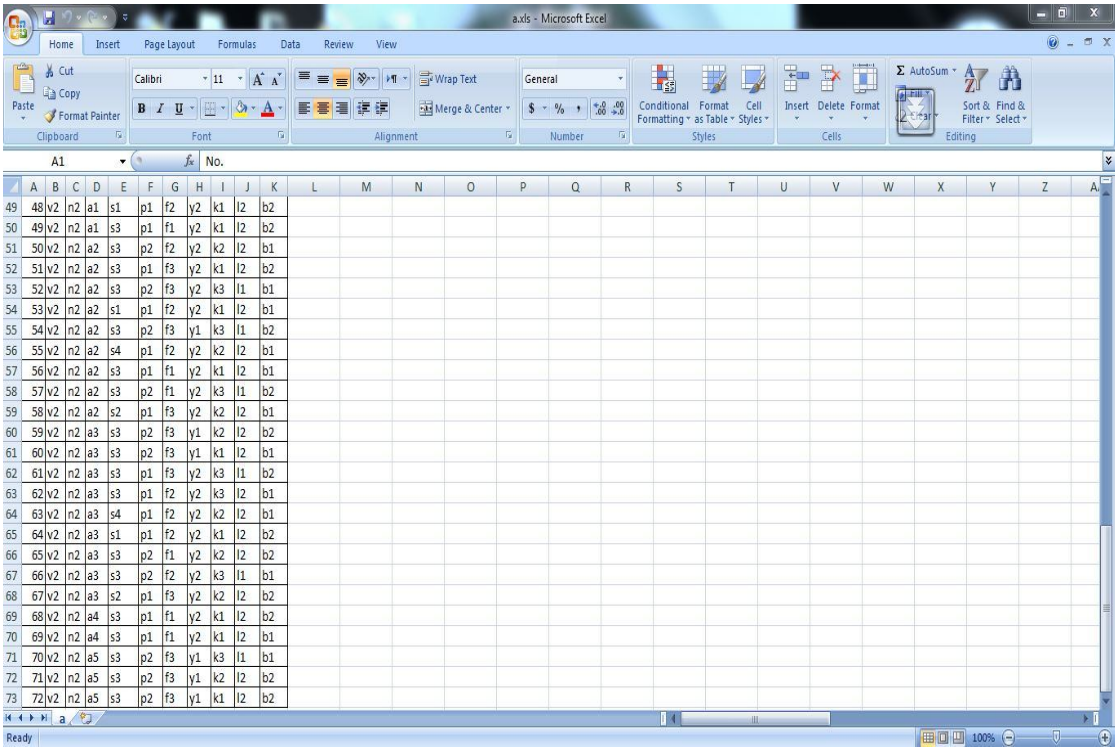


Figure (3-15): new test cases

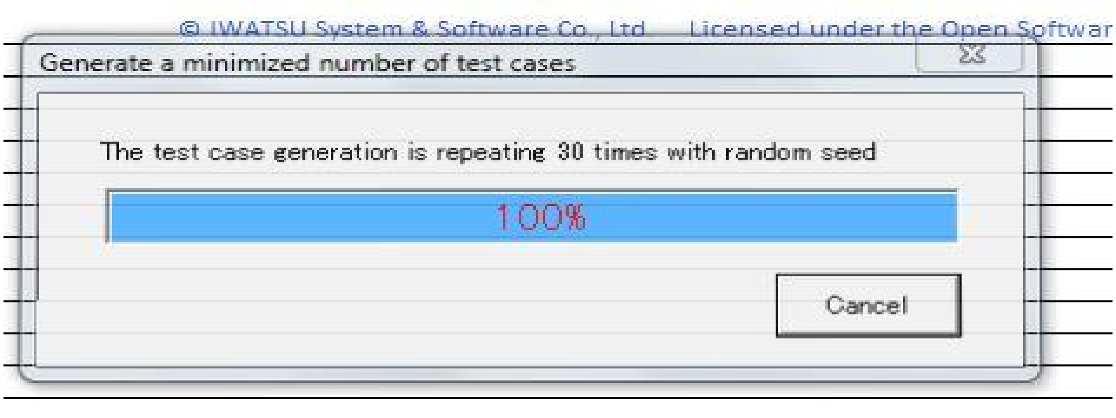


Figure (3-16): Progress Bar while minimizing the number of test cases

Displayedthe statistical informationFigure (3-17).

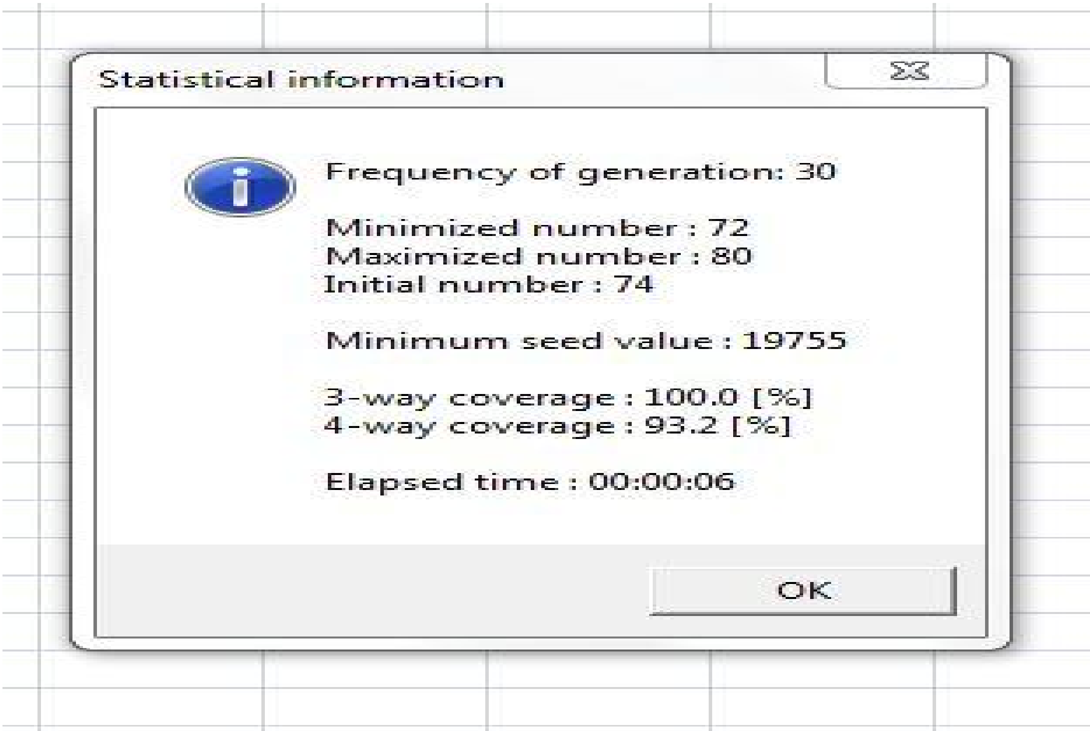


Figure (3-17): statistical information window

Then new model fileshown in Notepad format (a.txt)Figure (3-18).

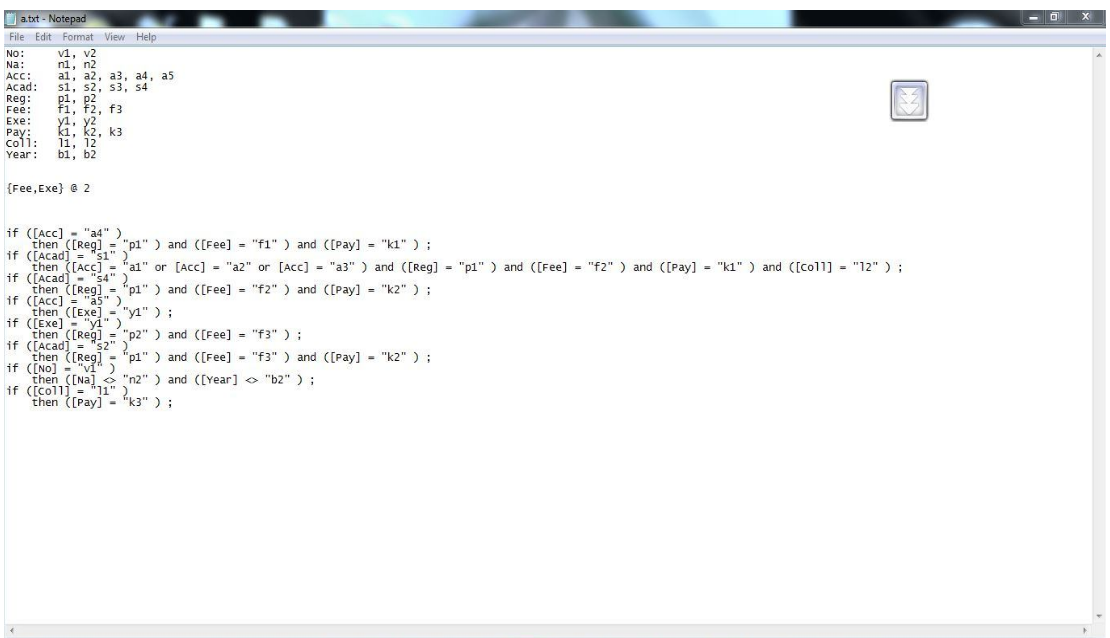


Figure (3-18):Notepad format

To generate the test cases for a combination of certain parameters that is different from the "Number of combined parameters" in the environment settings form, used“extended sub-models" in the settings formFigure (3-19).Chased (exemption and tuition fees) again with 3 "Number of combinedcertain parameters"Figure (3-20).The test cases can be created only for the particular parameters without having to significantly increase the number of test cases, Compared with the usual sub-model, the increase in the number of test cases can be drastically reduced.in my case the number Decreased from72 in the last result to 24 test casesFigure (3-21),Figure (3-22),Figure (3-23).

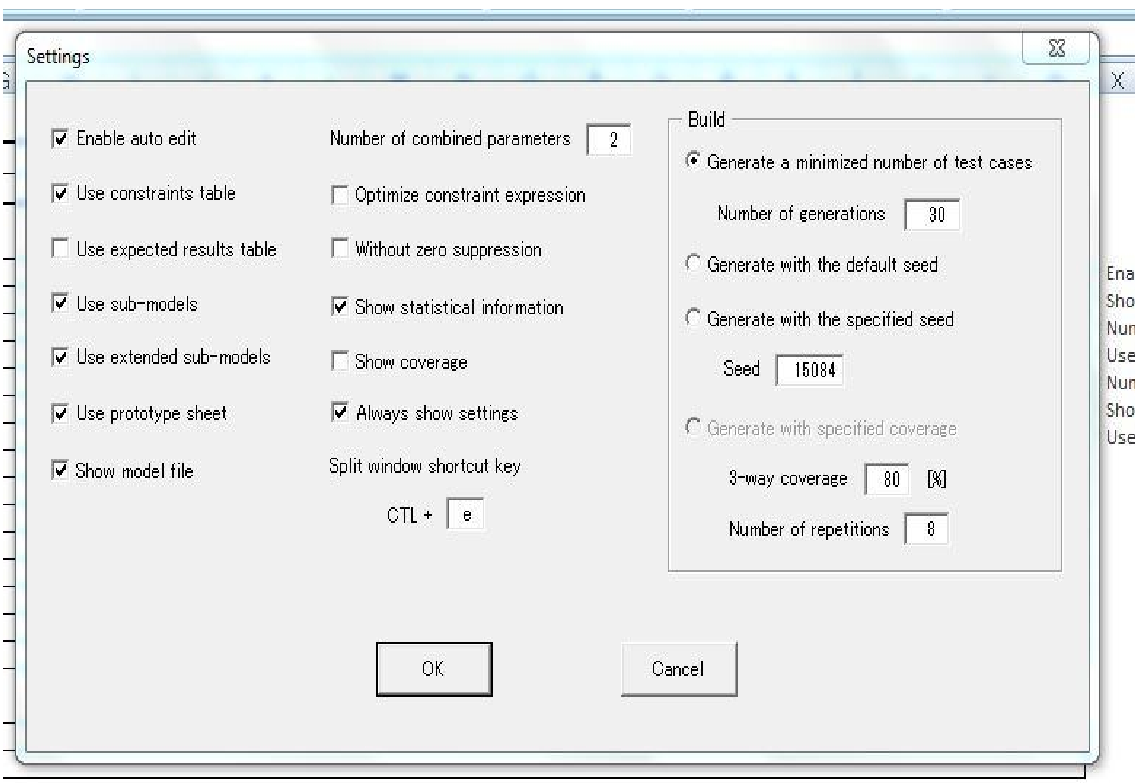


Figure (3-19): extended sub-models setting window

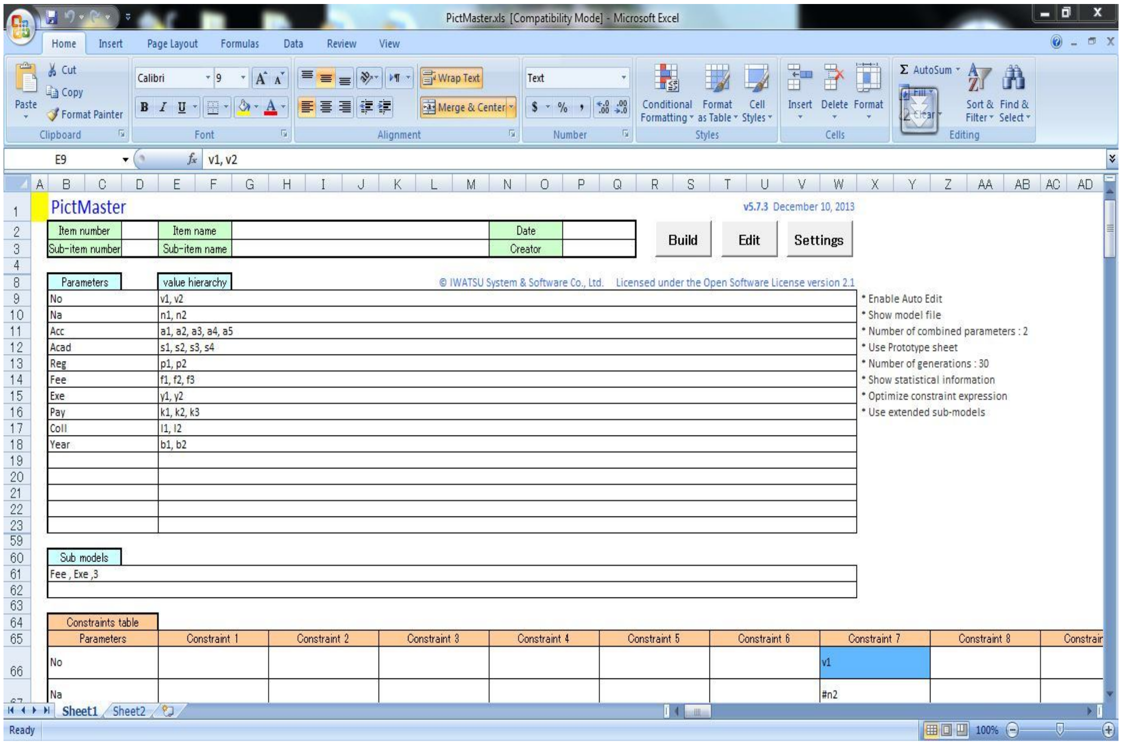


Figure (3-20): extended sub-models table

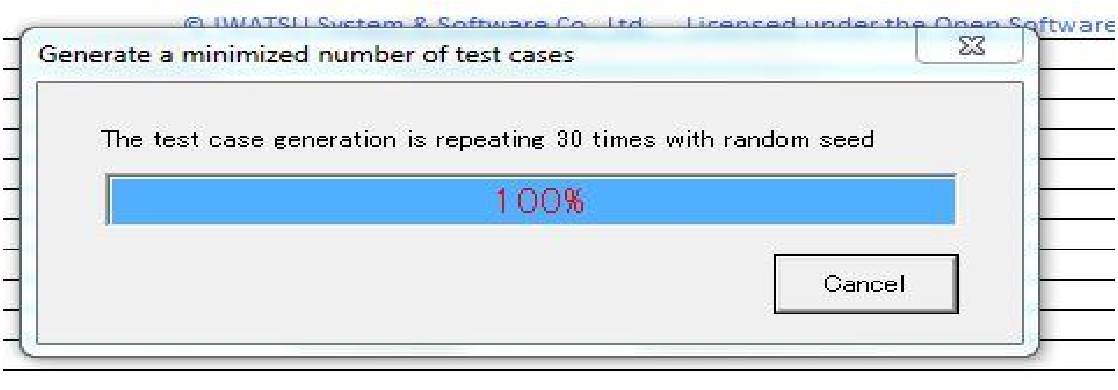
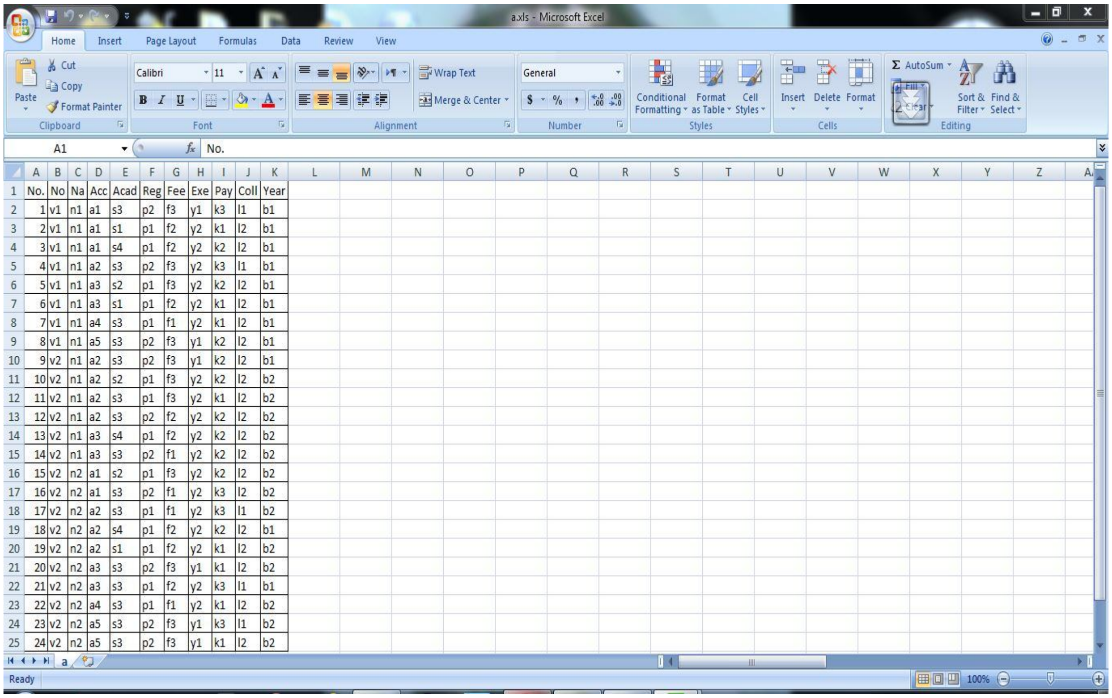


Figure (3-21): Progress Bar while minimizing the number of test cases after press build bottom



No.	No	Na	Acc	Acad	Reg	Fee	Exe	Pay	Coll	Year
1	v1	n1	a1	s3	p2	f3	y1	k3	l1	b1
2	v1	n1	a1	s1	p1	f2	y2	k1	l2	b1
3	v1	n1	a1	s4	p1	f2	y2	k2	l2	b1
4	v1	n1	a2	s3	p2	f3	y2	k3	l1	b1
5	v1	n1	a3	s2	p1	f3	y2	k2	l2	b1
6	v1	n1	a3	s1	p1	f2	y2	k1	l2	b1
7	v1	n1	a4	s3	p1	f1	y2	k1	l2	b1
8	v1	n1	a5	s3	p2	f3	y1	k2	l2	b1
9	v2	n1	a2	s3	p2	f3	y1	k2	l2	b1
10	v2	n1	a2	s2	p1	f3	y2	k2	l2	b2
11	v2	n1	a2	s3	p1	f3	y2	k1	l2	b2
12	v2	n1	a2	s3	p2	f2	y2	k2	l2	b2
13	v2	n1	a3	s4	p1	f2	y2	k2	l2	b2
14	v2	n1	a3	s3	p2	f1	y2	k2	l2	b2
15	v2	n2	a1	s2	p1	f3	y2	k2	l2	b2
16	v2	n2	a1	s3	p2	f1	y2	k3	l2	b2
17	v2	n2	a2	s3	p1	f1	y2	k3	l1	b2
18	v2	n2	a2	s4	p1	f2	y2	k2	l2	b1
19	v2	n2	a2	s1	p1	f2	y2	k1	l2	b2
20	v2	n2	a3	s3	p2	f3	y1	k1	l2	b2
21	v2	n2	a3	s3	p1	f2	y2	k3	l1	b1
22	v2	n2	a4	s3	p1	f1	y2	k1	l2	b2
23	v2	n2	a5	s3	p2	f3	y1	k3	l1	b2
24	v2	n2	a5	s3	p2	f3	y1	k1	l2	b2

Figure (3-22): new test cases (24)

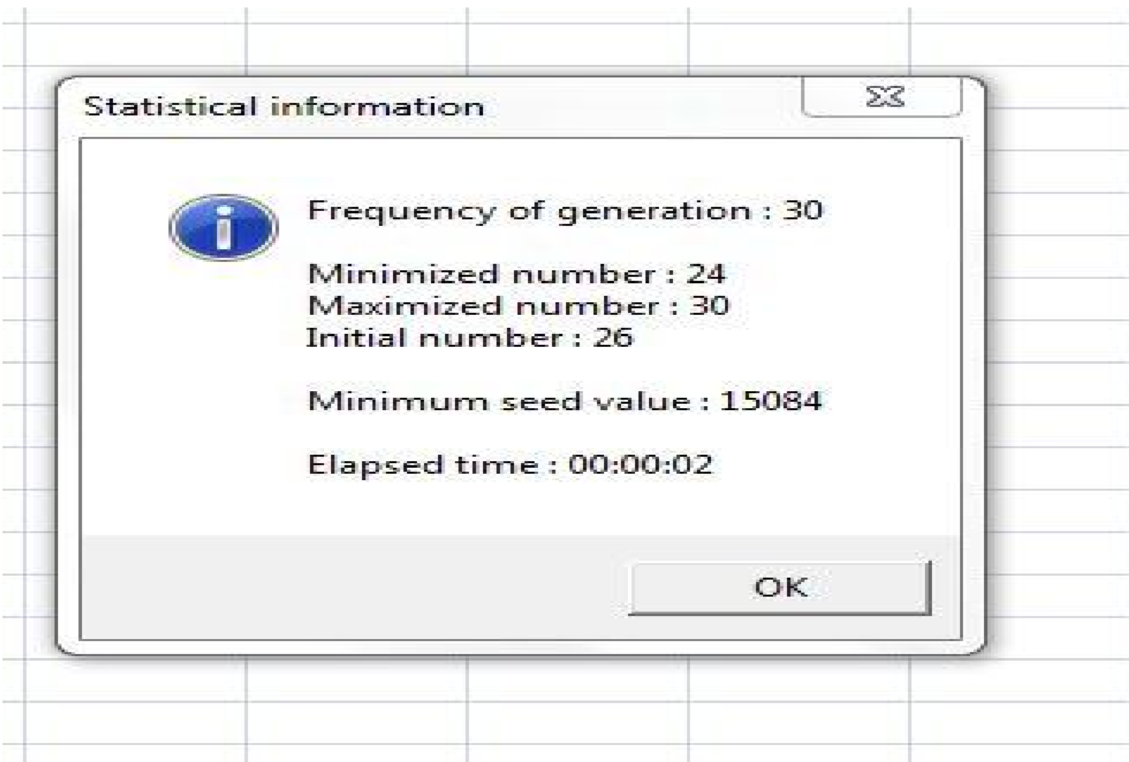


Figure (3-23): statistical information window

For Generating Test Cases to Ensure the Desired Coverage I specified from three-way to six-way coverage with the feature that generates the test cases. The t-way coverage where the value n + 1 (n specified in "number of combined parameters"

in the settings form) is ensured. For example, when you specified 2 in "Number of combined parameters," two-way coverage ensures 100 percent and three-way coverage is ensured with the value in "Desired coverage" Figure (3-24),Figure (3-25),Figure (3-26). The other t-way coverage shone inFigure (3-27) toFigure (3-28).

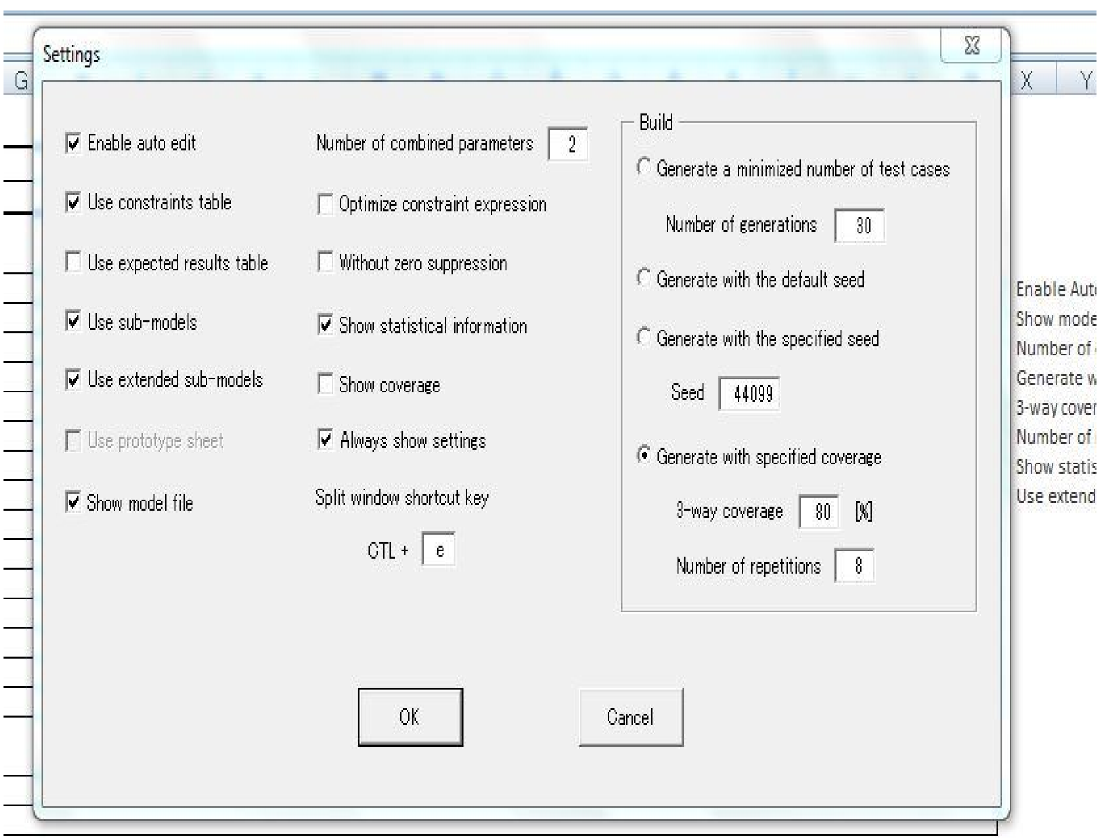


Figure (3-24): Desired coverage (3-way) setting window

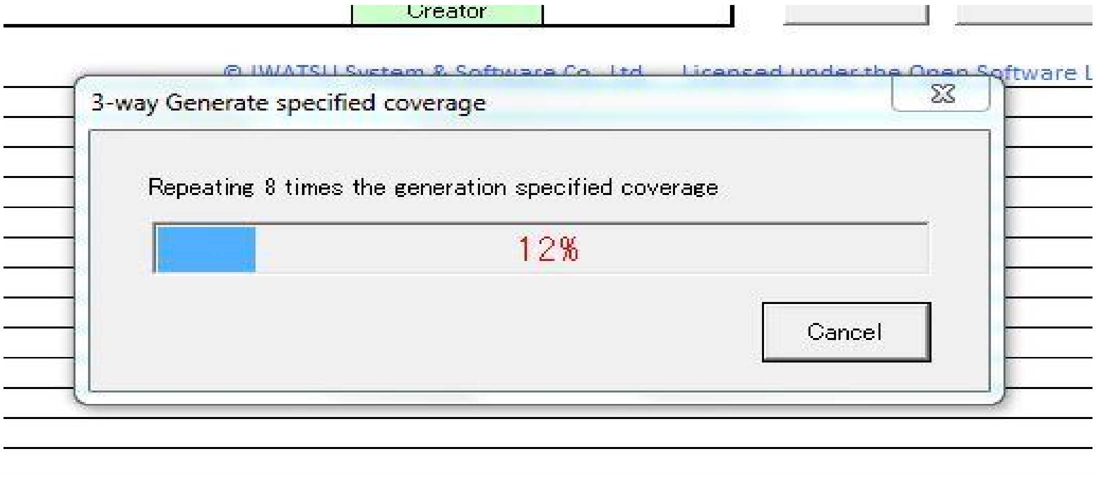
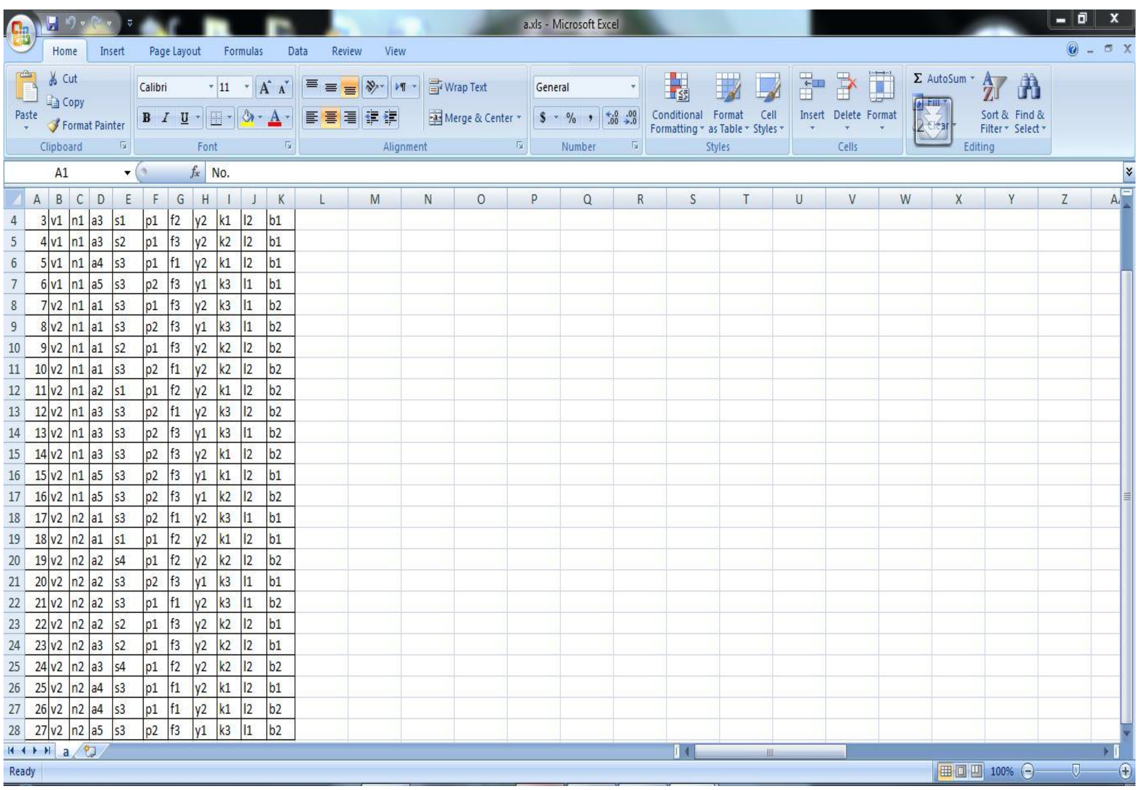


Figure (3-25): progress bar during generation with specified coverage



	A	B	C	D	E	F	G	H	I	J	K
4	3	v1	n1	a3	s1	p1	f2	y2	k1	l2	b1
5	4	v1	n1	a3	s2	p1	f3	y2	k2	l2	b1
6	5	v1	n1	a4	s3	p1	f1	y2	k1	l2	b1
7	6	v1	n1	a5	s3	p2	f3	y1	k3	l1	b1
8	7	v2	n1	a1	s3	p1	f3	y2	k3	l1	b2
9	8	v2	n1	a1	s3	p2	f3	y1	k3	l1	b2
10	9	v2	n1	a1	s2	p1	f3	y2	k2	l2	b2
11	10	v2	n1	a1	s3	p2	f1	y2	k2	l2	b2
12	11	v2	n1	a2	s1	p1	f2	y2	k1	l2	b2
13	12	v2	n1	a3	s3	p2	f1	y2	k3	l2	b2
14	13	v2	n1	a3	s3	p2	f3	y1	k3	l1	b2
15	14	v2	n1	a3	s3	p2	f3	y2	k1	l2	b2
16	15	v2	n1	a5	s3	p2	f3	y1	k1	l2	b1
17	16	v2	n1	a5	s3	p2	f3	y1	k2	l2	b2
18	17	v2	n2	a1	s3	p2	f1	y2	k3	l1	b1
19	18	v2	n2	a1	s1	p1	f2	y2	k1	l2	b1
20	19	v2	n2	a2	s4	p1	f2	y2	k2	l2	b2
21	20	v2	n2	a2	s3	p2	f3	y1	k3	l1	b1
22	21	v2	n2	a2	s3	p1	f1	y2	k3	l1	b2
23	22	v2	n2	a2	s2	p1	f3	y2	k2	l2	b1
24	23	v2	n2	a3	s2	p1	f3	y2	k2	l2	b1
25	24	v2	n2	a3	s4	p1	f2	y2	k2	l2	b2
26	25	v2	n2	a4	s3	p1	f1	y2	k1	l2	b1
27	26	v2	n2	a4	s3	p1	f1	y2	k1	l2	b2
28	27	v2	n2	a5	s3	p2	f3	y1	k3	l1	b2

Figure (3-26): new test cases (27)

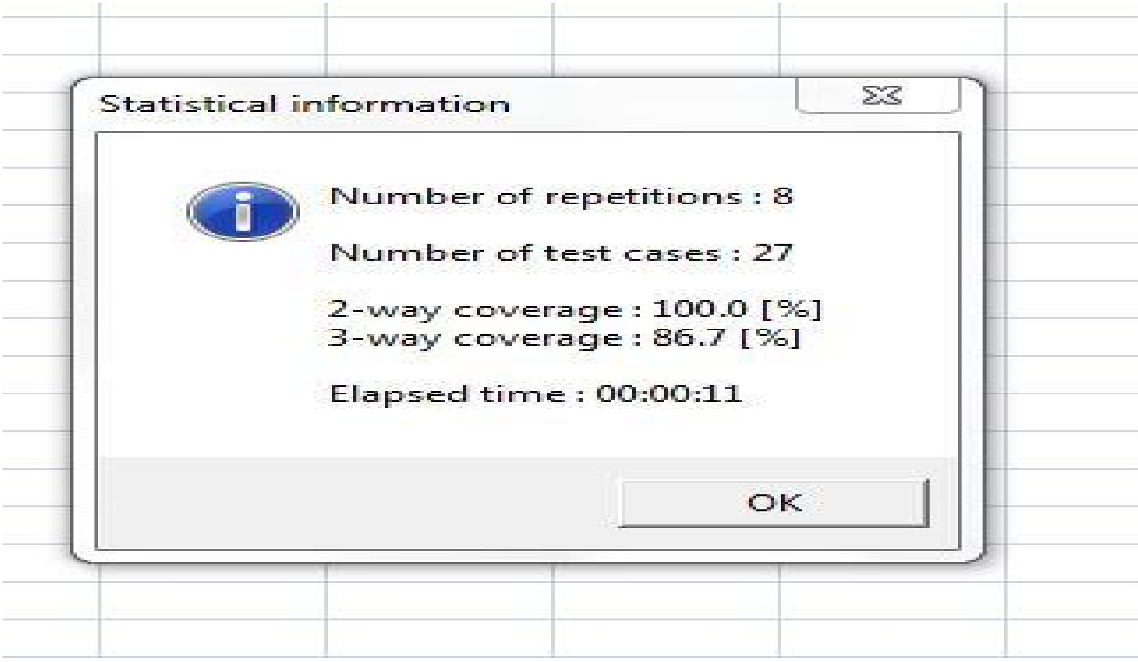


Figure (3-27): statistical information window

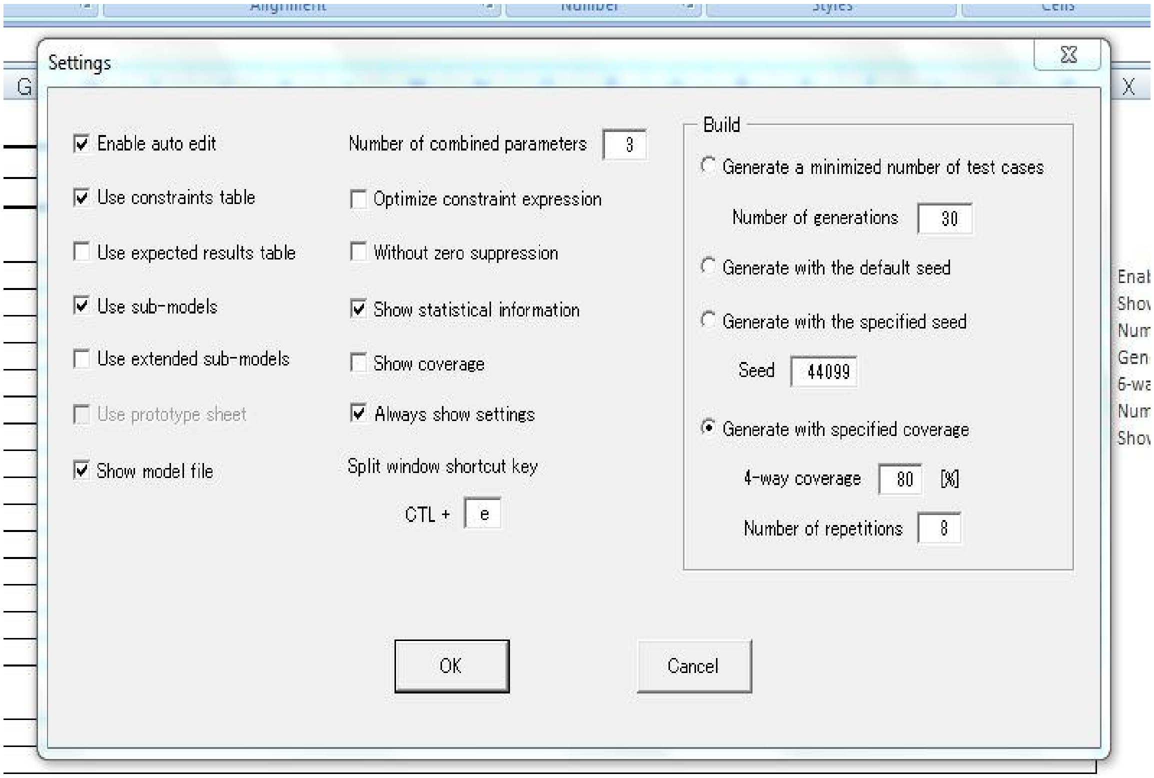


Figure (3-28): Desired coverage (4-way) setting window

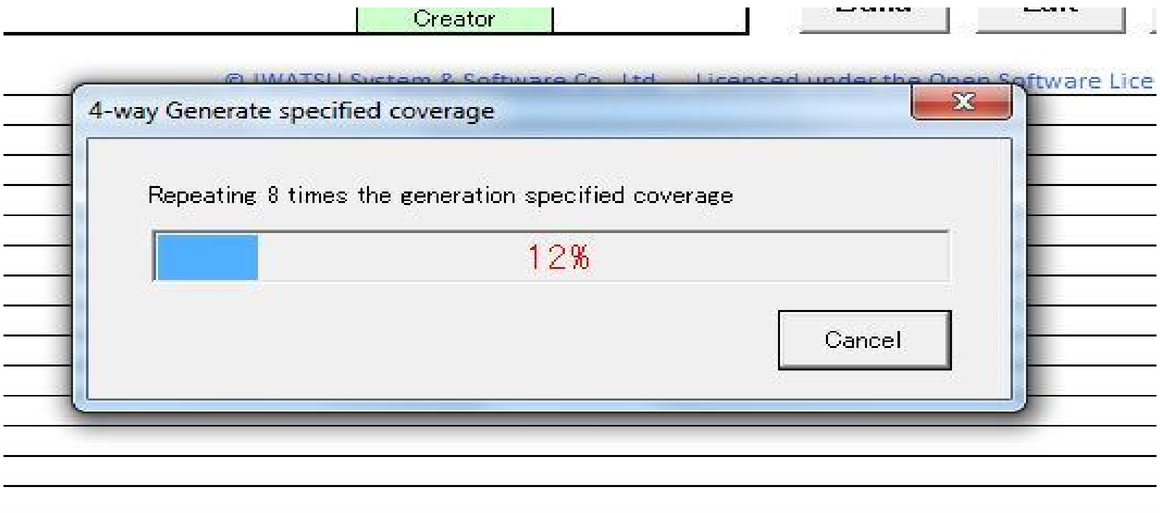


Figure (3-29): progress bar during generation with specified coverage

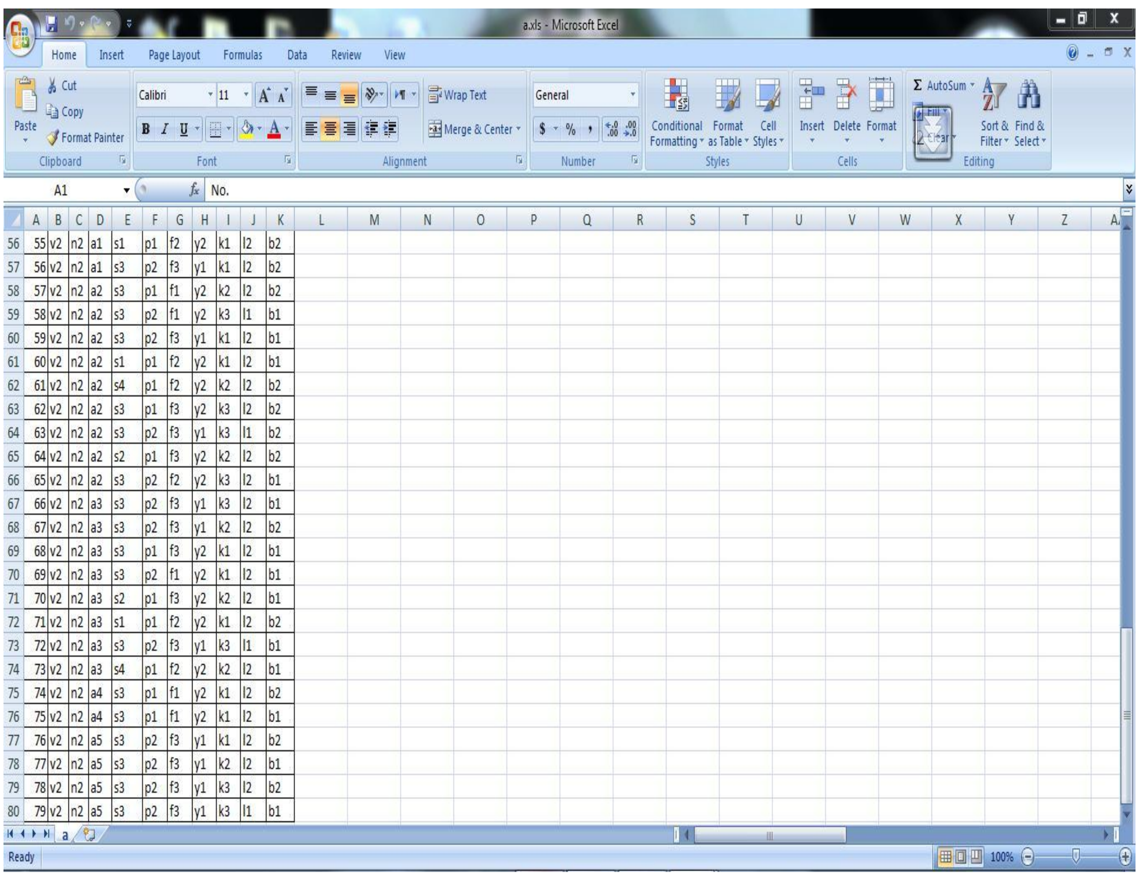


Figure (3-30): new test cases (79)

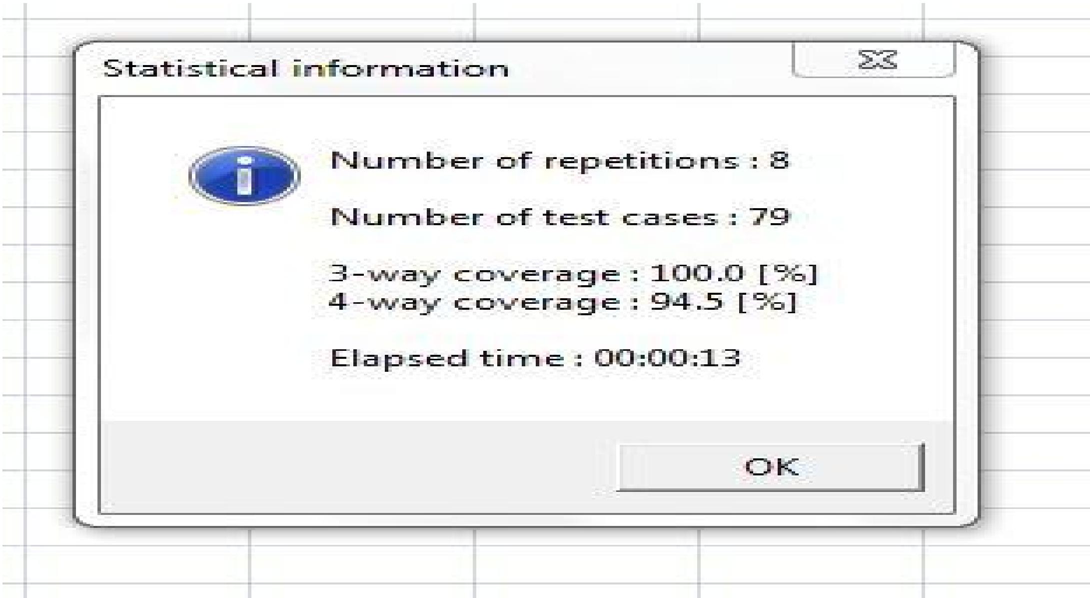
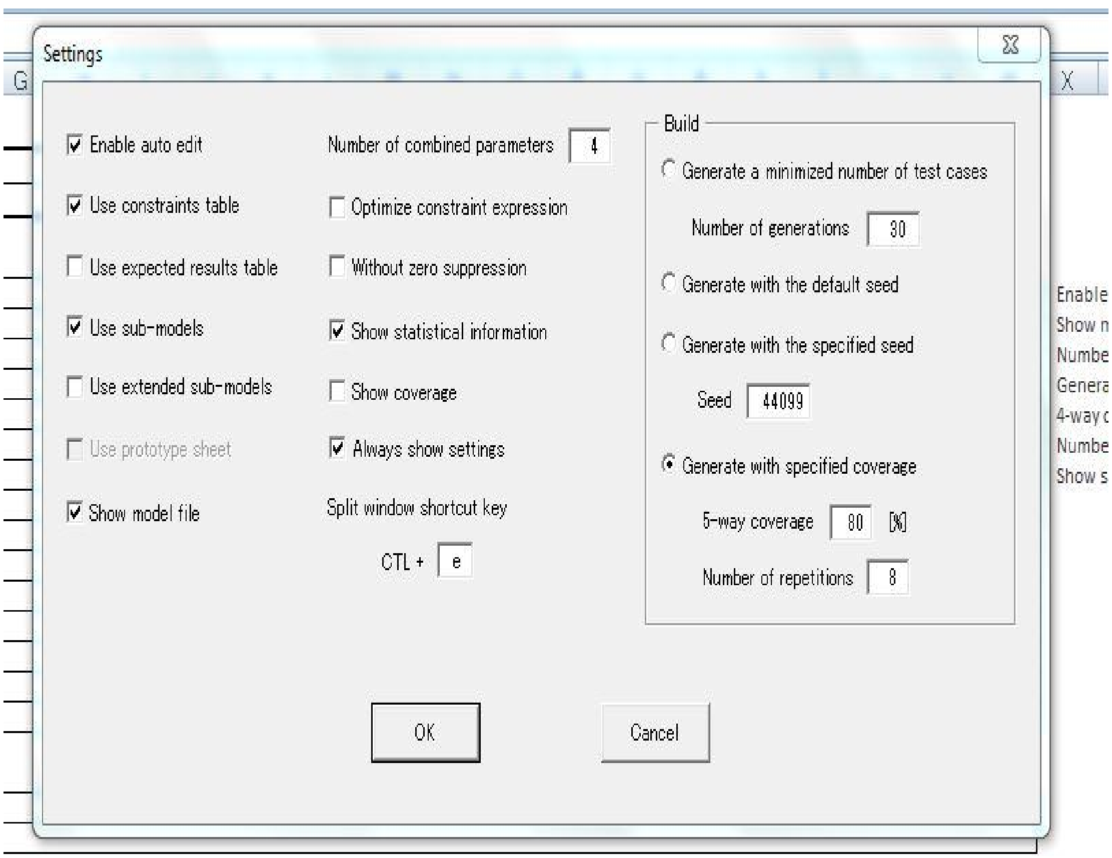


Figure (3-31): statistical information window



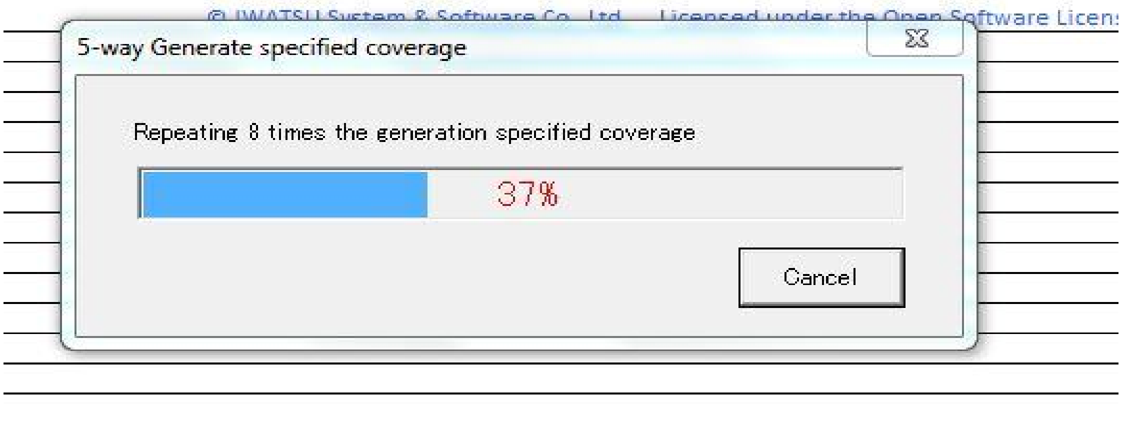
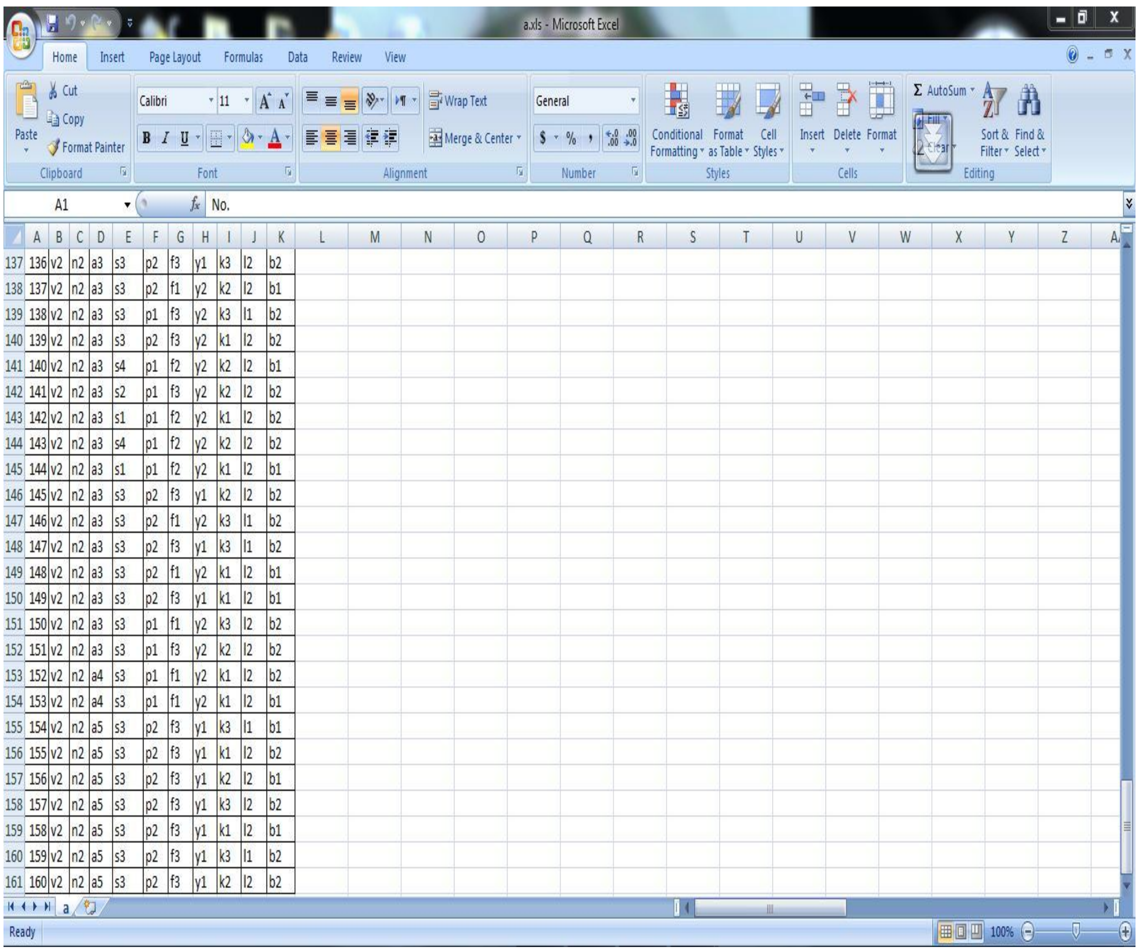


Figure (3-33): progress bar during generation with specified coverage



	A	B	C	D	E	F	G	H	I	J	K
137	136	v2	n2	a3	s3	p2	f3	y1	k3	l2	b2
138	137	v2	n2	a3	s3	p2	f1	y2	k2	l2	b1
139	138	v2	n2	a3	s3	p1	f3	y2	k3	l1	b2
140	139	v2	n2	a3	s3	p2	f3	y2	k1	l2	b2
141	140	v2	n2	a3	s4	p1	f2	y2	k2	l2	b1
142	141	v2	n2	a3	s2	p1	f3	y2	k2	l2	b2
143	142	v2	n2	a3	s1	p1	f2	y2	k1	l2	b2
144	143	v2	n2	a3	s4	p1	f2	y2	k2	l2	b2
145	144	v2	n2	a3	s1	p1	f2	y2	k1	l2	b1
146	145	v2	n2	a3	s3	p2	f3	y1	k2	l2	b2
147	146	v2	n2	a3	s3	p2	f1	y2	k3	l1	b2
148	147	v2	n2	a3	s3	p2	f3	y1	k3	l1	b2
149	148	v2	n2	a3	s3	p2	f1	y2	k1	l2	b1
150	149	v2	n2	a3	s3	p2	f3	y1	k1	l2	b1
151	150	v2	n2	a3	s3	p1	f1	y2	k3	l2	b2
152	151	v2	n2	a3	s3	p1	f3	y2	k2	l2	b2
153	152	v2	n2	a4	s3	p1	f1	y2	k1	l2	b2
154	153	v2	n2	a4	s3	p1	f1	y2	k1	l2	b1
155	154	v2	n2	a5	s3	p2	f3	y1	k3	l1	b1
156	155	v2	n2	a5	s3	p2	f3	y1	k1	l2	b2
157	156	v2	n2	a5	s3	p2	f3	y1	k2	l2	b1
158	157	v2	n2	a5	s3	p2	f3	y1	k3	l2	b2
159	158	v2	n2	a5	s3	p2	f3	y1	k1	l2	b1
160	159	v2	n2	a5	s3	p2	f3	y1	k3	l1	b2
161	160	v2	n2	a5	s3	p2	f3	y1	k2	l2	b2

Figure (3-34): new test cases (160)

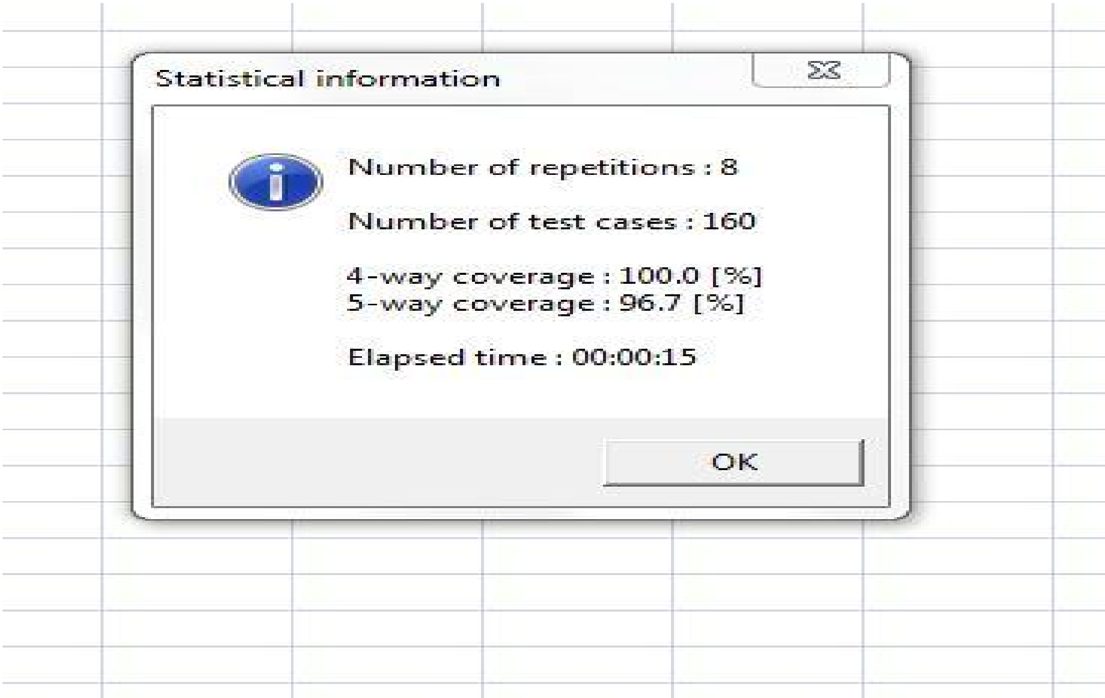


Figure (3-35): statistical information window

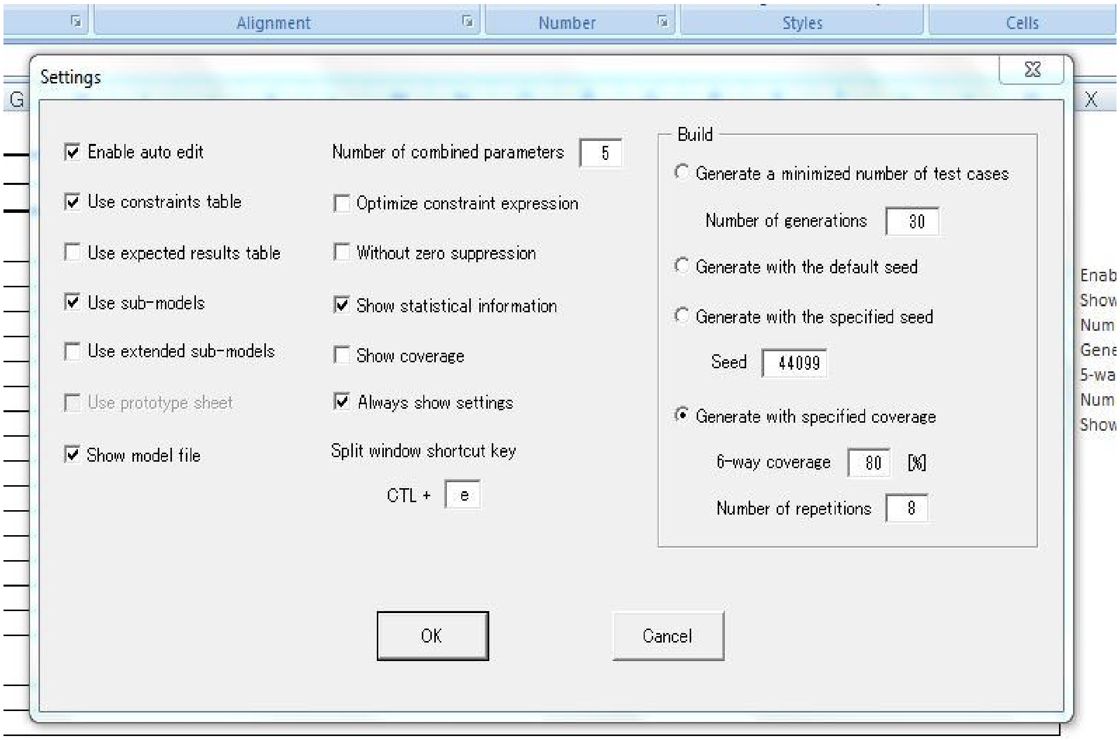


Figure (3-36): Desired coverage (6-way) setting window

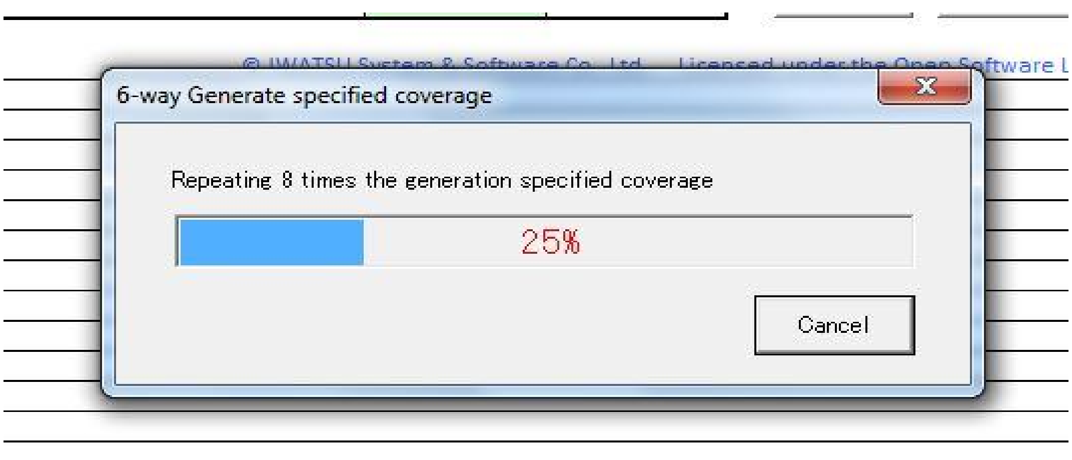
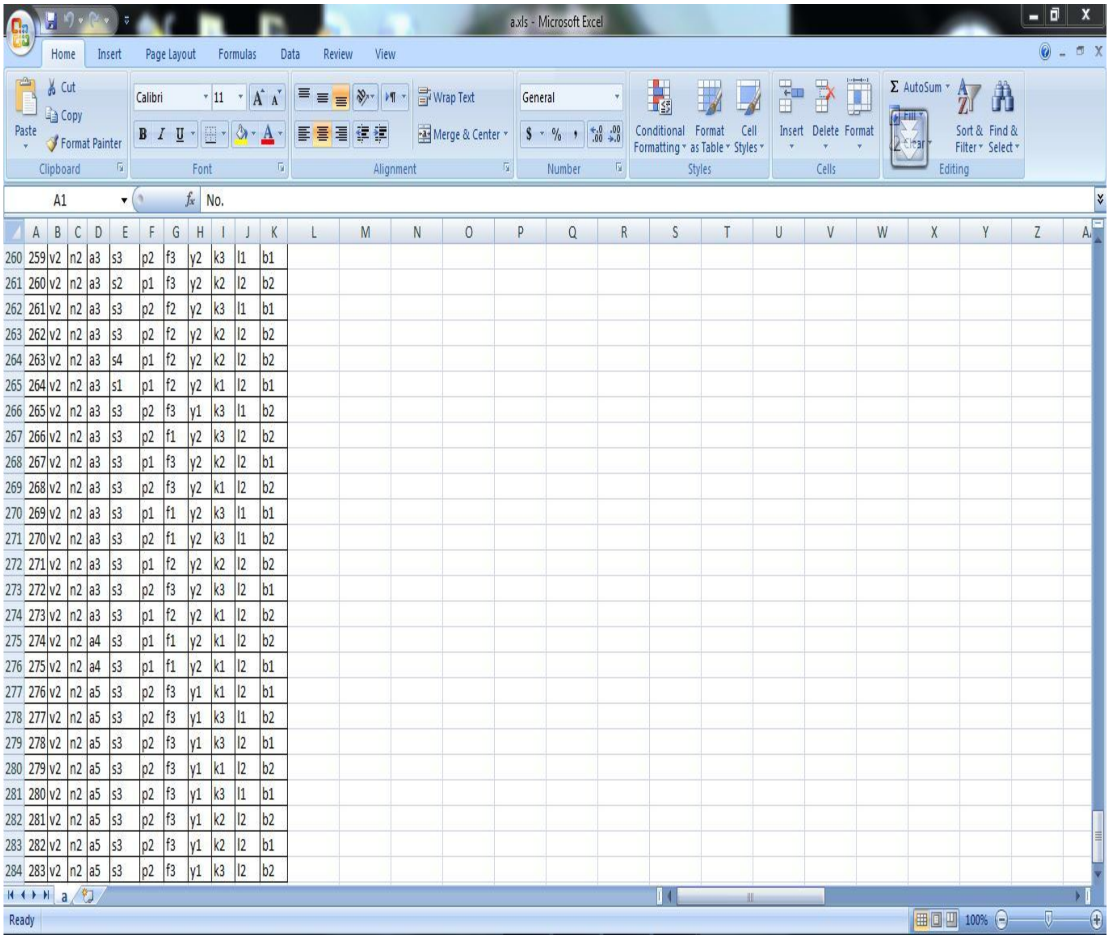


Figure (3-37): progress bar during generation with specified coverage



	A	B	C	D	E	F	G	H	I	J	K
260	259	v2	n2	a3	s3	p2	f3	y2	k3	l1	b1
261	260	v2	n2	a3	s2	p1	f3	y2	k2	l2	b2
262	261	v2	n2	a3	s3	p2	f2	y2	k3	l1	b1
263	262	v2	n2	a3	s3	p2	f2	y2	k2	l2	b2
264	263	v2	n2	a3	s4	p1	f2	y2	k2	l2	b2
265	264	v2	n2	a3	s1	p1	f2	y2	k1	l2	b1
266	265	v2	n2	a3	s3	p2	f3	y1	k3	l1	b2
267	266	v2	n2	a3	s3	p2	f1	y2	k3	l2	b2
268	267	v2	n2	a3	s3	p1	f3	y2	k2	l2	b1
269	268	v2	n2	a3	s3	p2	f3	y2	k1	l2	b2
270	269	v2	n2	a3	s3	p1	f1	y2	k3	l1	b1
271	270	v2	n2	a3	s3	p2	f1	y2	k3	l1	b2
272	271	v2	n2	a3	s3	p1	f2	y2	k2	l2	b2
273	272	v2	n2	a3	s3	p2	f3	y2	k3	l2	b1
274	273	v2	n2	a3	s3	p1	f2	y2	k1	l2	b2
275	274	v2	n2	a4	s3	p1	f1	y2	k1	l2	b2
276	275	v2	n2	a4	s3	p1	f1	y2	k1	l2	b1
277	276	v2	n2	a5	s3	p2	f3	y1	k1	l2	b1
278	277	v2	n2	a5	s3	p2	f3	y1	k3	l1	b2
279	278	v2	n2	a5	s3	p2	f3	y1	k3	l2	b1
280	279	v2	n2	a5	s3	p2	f3	y1	k1	l2	b2
281	280	v2	n2	a5	s3	p2	f3	y1	k3	l1	b1
282	281	v2	n2	a5	s3	p2	f3	y1	k2	l2	b2
283	282	v2	n2	a5	s3	p2	f3	y1	k2	l2	b1
284	283	v2	n2	a5	s3	p2	f3	y1	k3	l2	b2

Figure (3-38): new test cases (283)

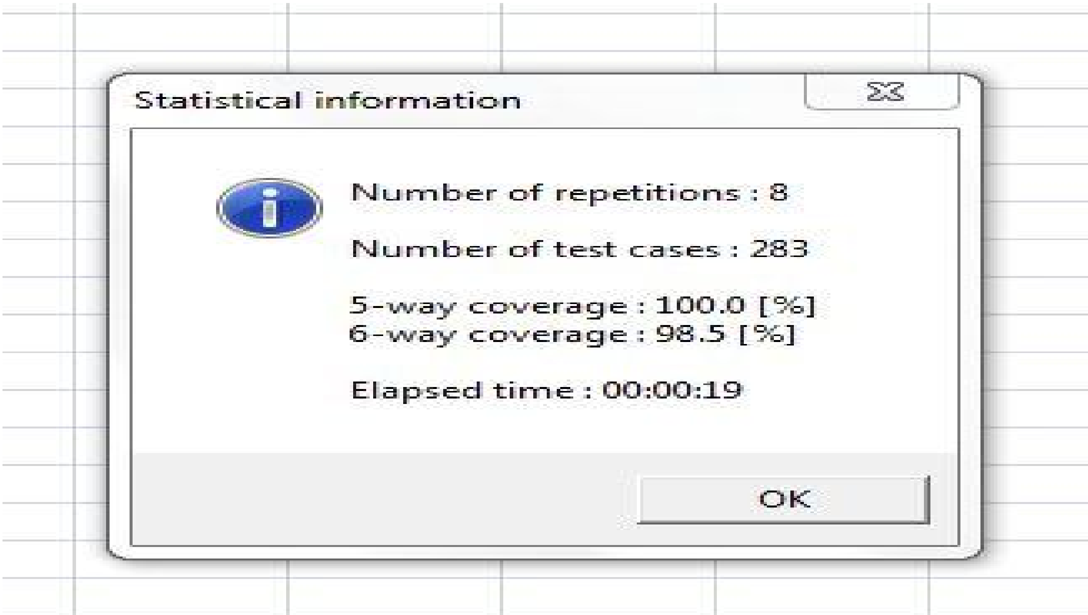


Figure (3-39): statistical information window

The Summary in table(3-4):

Table (3-4): summary

t-way coverage	Number of Test cases	%
2	24	100
3	27	86.7
4	79	94.9
5	160	96.7
6	283	98.5

3.1.3.1GetExpected results

In this step, I identified the rules to get the expected results.

Rules:

- Students are allowed to register if his own (University ID, Name, Academic year) right and must have his academic (success or freeze).
- A student is not allowed to register if disturbed any of the conditions previously.

I used expected results tableFigure (3-40).

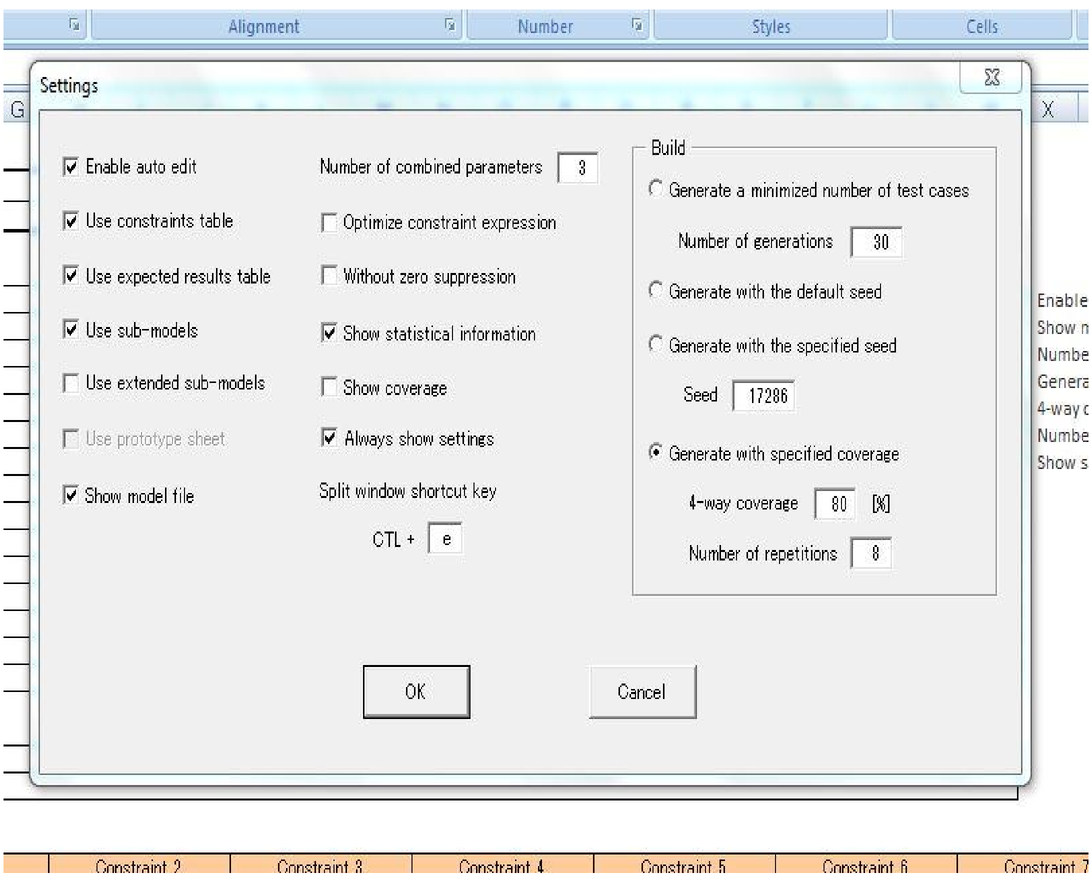
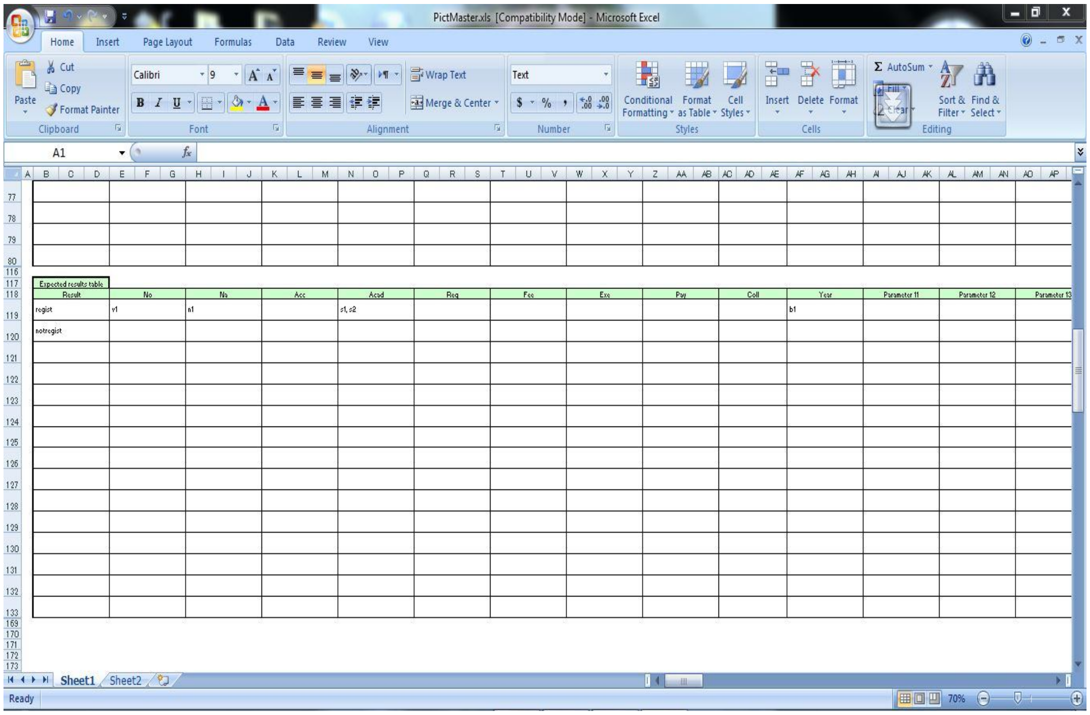


Figure (3-40): expected results setting window

Then entered the rules in theexpected results tableFigure (3-41).

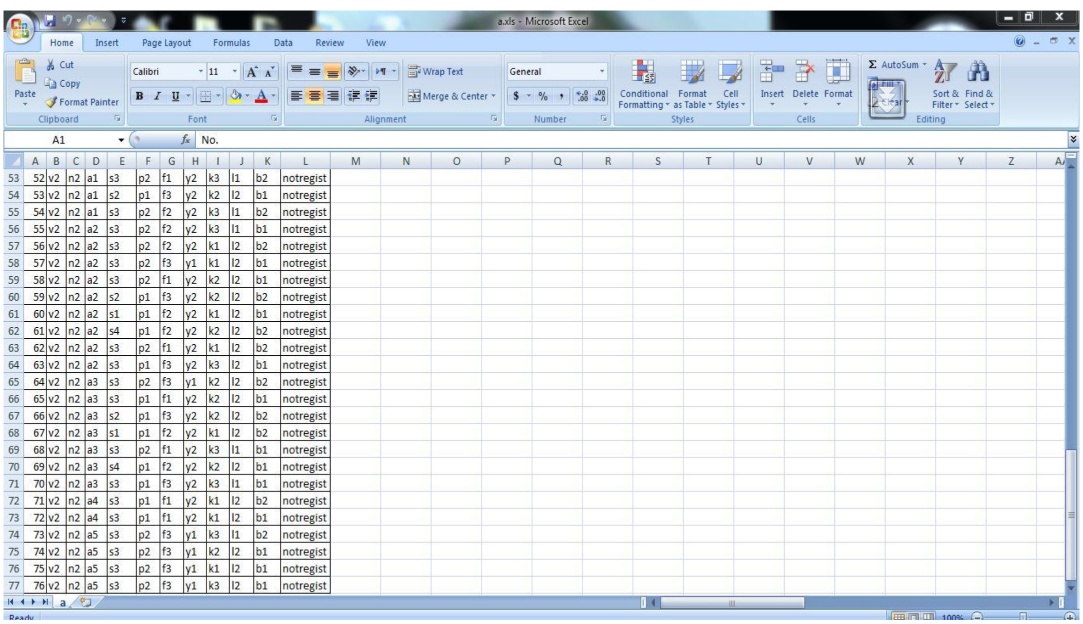


Expected results table

Result	No	Ns	Acc	Acsd	Req	Fee	Exe	Pay	Coll	Year	Parameter 11	Parameter 12	Parameter 13
regist	v1	n1		s1, s2						b1			
notregist													

Figure (3-41): expected results table

I used build bottom to get the resultFigure (3-42), (3-43).



	A	B	C	D	E	F	G	H	I	J	K	L
53	52	v2	n2	a1	s3	p2	f1	y2	k3	l1	b2	notregist
54	53	v2	n2	a1	s2	p1	f3	y2	k2	l2	b1	notregist
55	54	v2	n2	a1	s3	p2	f2	y2	k3	l1	b2	notregist
56	55	v2	n2	a2	s3	p2	f2	y2	k3	l1	b1	notregist
57	56	v2	n2	a2	s3	p2	f2	y2	k1	l2	b2	notregist
58	57	v2	n2	a2	s3	p2	f3	y1	k1	l2	b1	notregist
59	58	v2	n2	a2	s3	p2	f1	y2	k2	l2	b1	notregist
60	59	v2	n2	a2	s2	p1	f3	y2	k2	l2	b2	notregist
61	60	v2	n2	a2	s1	p1	f2	y2	k1	l2	b1	notregist
62	61	v2	n2	a2	s4	p1	f2	y2	k2	l2	b2	notregist
63	62	v2	n2	a2	s3	p2	f1	y2	k1	l2	b2	notregist
64	63	v2	n2	a2	s3	p1	f3	y2	k3	l2	b1	notregist
65	64	v2	n2	a3	s3	p2	f3	y1	k2	l2	b2	notregist
66	65	v2	n2	a3	s3	p1	f1	y2	k2	l2	b1	notregist
67	66	v2	n2	a3	s2	p1	f3	y2	k2	l2	b2	notregist
68	67	v2	n2	a3	s1	p1	f2	y2	k1	l2	b2	notregist
69	68	v2	n2	a3	s3	p2	f1	y2	k3	l1	b1	notregist
70	69	v2	n2	a3	s4	p1	f2	y2	k2	l2	b1	notregist
71	70	v2	n2	a3	s3	p1	f3	y2	k3	l1	b1	notregist
72	71	v2	n2	a4	s3	p1	f1	y2	k1	l2	b2	notregist
73	72	v2	n2	a4	s3	p1	f1	y2	k1	l2	b1	notregist
74	73	v2	n2	a5	s3	p2	f3	y1	k3	l1	b2	notregist
75	74	v2	n2	a5	s3	p2	f3	y1	k2	l2	b1	notregist
76	75	v2	n2	a5	s3	p2	f3	y1	k1	l2	b1	notregist
77	76	v2	n2	a5	s3	p2	f3	y1	k3	l2	b1	notregist

Figure (3-42):Expected results

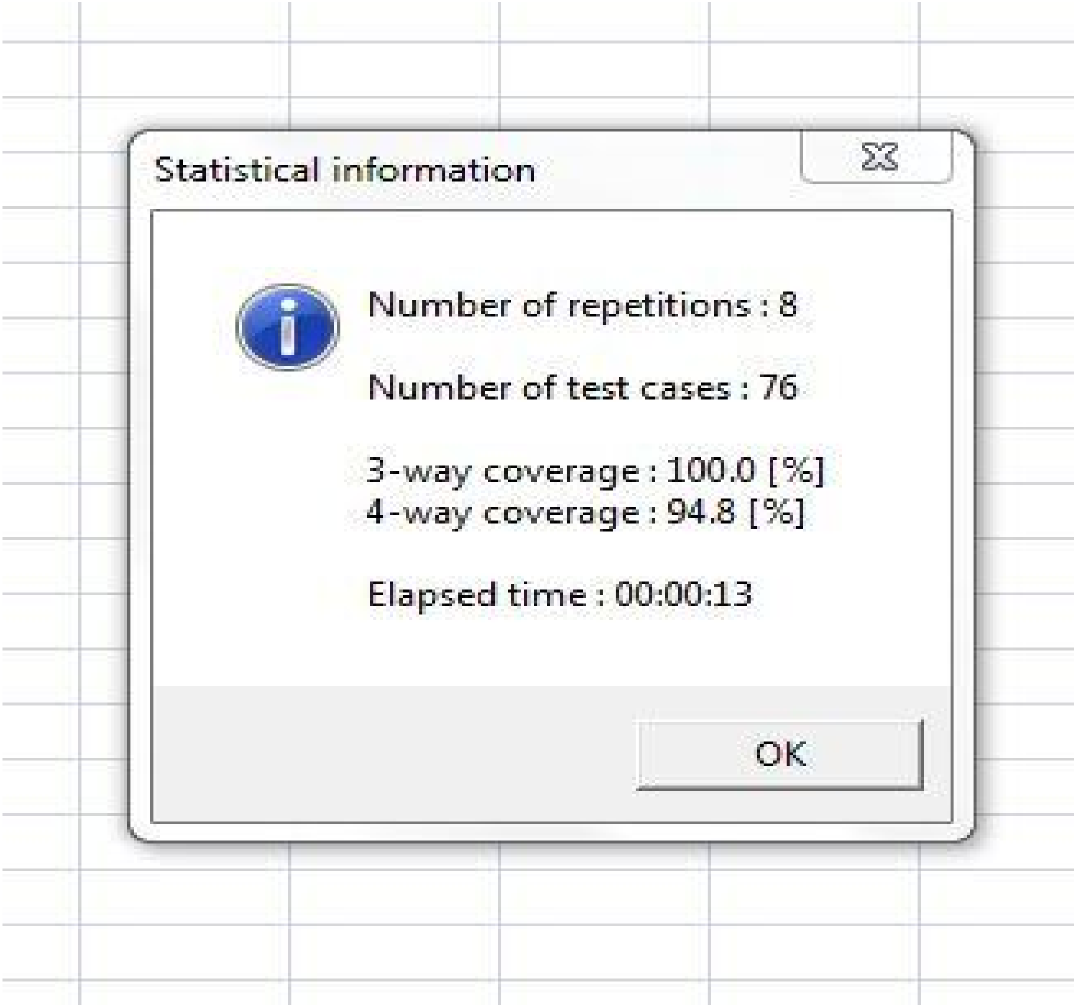


Figure (3-43): Statistical Information window

As the final result I get 76 test cases with 6 cases can register and 70 case not register.

CHAPTER 4

Analysis of Result

This research is challenge to support the useof Combinatorial Testing in Practice. PictMaster tool generates test casesafter it is give the model. Asthe taster could controls the results by specifying constraints, sub models, extend sub models desired coverage and rules.

Upon checking in the final outcomes obtained in the experiments which were conducted in this versiontest casesfrom (no.28 to no.76) contained invalid values of (University ID, Name, The academic year),these test cases not werevery important because they were predetermined in the basic system requirements, so selected test no. 28 to test the invalid University ID withother parameterFor more verification.

The expected results explained that not any of the previous cases could complete the registration process and that was what must be achieved in the system. Test cases no.1 to no.27 explained that 6 cases can complete the registration process and 21 cases can't, althoughvalidity of University ID, Name and Academic Year were correct. Finally the total number of the test cases this research focused upon decreased from76to 28.

To analyze these results within the field of black box testing the researcher used User Testing Dominant Style (Kaner, 2003),the researcher selected nine ofRegistersof colleges .Then the researcher gave each of them the test cases designed in such detail as in Figure (4-1)to execute and report whether the program passed or failed.

No.	University ID	Name	Type Acceptance	Academic position	Registration fee	Tuition fees	Exception	Payment type	college	Academic year	Expected Result
1	Correct Number	Correct name	general	Success	Specific	Half fees	Non -exempt	First premium	theoretical	Correct year	Regist
2	Correct Number	Correct name	general	Freeze	Specific	No fees	Non -exempt	last premium	theoretical	Correct year	Regist
3	Correct Number	Correct name	general	Role of the second	un specified	No fees	Exempt	First premium	theoretical	Correct year	Notregist
4	Correct Number	Correct name	general	Role of the second	un specified	No fees	Exempt	Other fees	scientific	Correct year	Notregist
5	Correct Number	Correct name	general	Role of the second	un specified	Full fees	Non -exempt	last premium	theoretical	Correct year	Notregist
6	Correct Number	Correct name	general	Role of the second	Specific	No fees	Non -exempt	First premium	theoretical	Correct year	Notregist
7	Correct Number	Correct name	general	Role of the second	un specified	Half fees	Non -exempt	Other fees	theoretical	Correct year	Notregist
8	Correct Number	Correct name	general	Repeat	specific	Half fees	Non -exempt	last premium	theoretical	Correct year	Notregist
9	Correct Number	Correct name	general	Role of the second	Specific	Half fees	Non -exempt	Other fees	scientific	Correct year	Notregist
10	Correct number	Correct name	general	Role of the second	un specified	No fees	Non -exempt	Other fees	scientific	Correct year	Notregist
11	Correct number	Correct name	private	Role of the second	un specified	Full fees	Non -exempt	Other fees	scientific	Correct year	Notregist
12	Correct number	Correct name	private	Freeze	Specific	No fees	Non -exempt	last premium	theoretical	Correct year	Regist
13	Correct number	Correct name	private	Success	Specific	Half fees	Non -exempt	First premium	theoretical	Correct year	Regist
14	Correct number	Correct name	private	Role of the second	un specified	No fees	Exempt	Other fees	scientific	Correct year	Notregist
15	Correct number	Correct name	private	Repeat	Specific	Half fees	Non -exempt	last premium	theoretical	Correct year	Notregist
16	Correct number	Correct name	Mature study	Role of the second	Specific	No fees	Non -exempt	Other fees	theoretical	Correct year	Notregist
17	Correct number	Correct name	Mature study	Success	Specific	Half fees	Non -exempt	First premium	theoretical	Correct year	Regist
18	Correct number	Correct name	Mature study	Repeat	Specific	Half fees	Non -exempt	last premium	theoretical	Correct year	Notregist
19	Correct number	Correct name	Mature study	Role of the second	un specified	No fees	Exempt	First premium	theoretical	Correct year	Notregist
20	Correct number	Correct name	Mature study	Role of the second	Specific	No fees	Non -exempt	Other fees	scientific	Correct year	Notregist
21	Correct number	Correct name	Mature study	Freeze	Specific	No fees	Non -exempt	last premium	theoretical	Correct year	Regist
22	Correct number	Correct name	Mature study	Role of the second	Specific	Full fees	Non -exempt	last premium	theoretical	Correct year	Notregist
23	Correct number	Correct name	external	Role of the second	Specific	Full fees	Non -exempt	first premium	theoretical	Correct year	Notregist
24	Correct number	Correct name	Darfur student	Role of the second	un specified	No fees	Exempt	last premium	theoretical	Correct year	Notregist
25	Correct number	Correct name	Darfur student	Role of the second	un specified	No fees	Exempt	Other fees	theoretical	Correct year	Notregist
26	Correct number	Correct name	Darfur student	Role of the second	un specified	No fees	Exempt	First premium	theoretical	Correct year	Notregist
27	Correct number	Correct name	Darfur student	Role of the second	un specified	No fees	Exempt	Other fees	scientific	Correct year	Notregist
28	Wrong number	Correct name	general	Role of the second	un specified	No fees	Non -exempt	last premium	theoretical	Correct year	Notregist

Figure (4-1): Test Report

Summary ofthe resultsreached by theRegisterscameas follows:

Table (4-1): Result

Test no.	result
1	**pass**
2	**pass**
3	**pass**
4	**pass**
5	**fail**
6	**fail**
7	**fail**
8	**fail**
9	**fail**
10	**pass**
11	**fail**
12	**pass**
13	**pass**
14	**pass**
15	**fail**
16	**fail**
17	**pass**
18	**fail**
19	**pass**
20	**fail**
21	**pass**
22	**fail**
23	**fail**
24	**pass**
25	**pass**
26	**pass**
27	**pass**
28	**pass**

After analyzing these resultsand comparing them with expected results shown in figure (3-42), concluded that the students that their position academic (role of the second or Repeat) allowed to register if any of the registration fees or tuition specific.

This is a major fault in the Registration of Omdurman Islamic university program and cannot be detected through regular testing.

This result have been achieved after many experiment were conducted using Pictmaster tool.

CHAPTER 5

Conclusion

Combinatorial Testing can detect failures triggered by interactions of parameters in the Software Under Test (SUT) with a covering array test suite generated by some sampling mechanisms. Combinatorial testing makes an excellent trade- off between test effort and test effectiveness.

This research presents a three-step approach to apply combinatorial testing. First the researcher create an abstract model for the system. Then, based on that model, a combinatorial abstract test set was generated. Then a set of concrete tests were driven from these abstract tests and applied combinatorial testing to mentioned program. The details of the abstract model and the results of applying combinatorial testing were presented in the research. The results show that combinatorial testing can detect faults of the ElectronicRegistration programs, and this is more effective than code testing.

This conclusion cannot be generalized to all other applications. The type of interaction is highly dependent on the problem at hand. It is the knowledge,understanding and the software tester that is crucial.

Suggestions for futureresearch

While much useful research work has been done in the last decade, the adoption of interaction testing for studying and testing real life systems has not been widespread. In order to address this issue, more research into the algorithms and techniques are required to facilitate its adoption in the main stream of software engineering.

In the future, the researcher plans to conduct more empirical studies on larger and more complex programs. believe this research will provide guidance for practitioners to apply combinatorial testing in practice.

REFERENCES

Alsewari,et al., 2012.A harmony search based pairwise sampling strategy for combinatorial testing [online] Available from:http://www.academicjournals.org/IJPS [accessed9 February, 2013].

Alton.B, et al., 2012.Effectiveness of pairwise testingfor software with Boolean inputs [online] Available from: core.ecu.edu/vilkomirs/Papers/Vilkomir-CT-2012.pd*f*[accessed 2013].

Automated Combinatorial Testing for Software (ACTS). [Online] Available from: http://www.nist.gov/itl/csd/scm/acts.cfm.

Bach, J. Schroeder, 2004. Pairwise Testing: A Best Practice That Isn't.[Online] Available from: www.testingeducation.org/wtst5/PairwisePNSQC2004.pdf.

Czerwonka, 2008. Pairwise Testing in the Real World: Practical Extensions to Test-Case Scenarios. [Online] Available from:http://msdn.microsoft.com/en-us/library/cc150619.aspx.[accessed February 2013].

Huima, 2012. Understanding Pairwise Test Generation. [online] Available from: http://www.conformiq.com/2012/01/understanding-pairwise-test-generation.[accessed 3 January 2013]**.**

Kaner, et al., 1999. Testing Computer Software, [online] Available from: http://en.wikipedia.org/wiki/Software_testing.

Kaner, et al., 2003. An introduction to scenario testing,[online] Available from: www.kaner.com/pdfs/ScenarioIntroVer4.pdf. [AccessedOctober, 2013]**.**

Kobayashi, et al., 2001.A new method for constructing pair-wise covering designs for softwaretesting. [Online]Available from:http://www.sciencedirect.com /science/article/pii/S002001900100195817. [Accessed March 2014].

Kotthoff, 2012. Algorithm Selection for Combinatorial Search Problems: A survey. [Online]Available from:http://scholar.google.com.[Accessed 30 Oct 2013].

Kuhn, et al., 2004. Software Fault Interactions and Implications for Software Testing,[online] Available from: http://dl.acm.org/citation.cfm?id=998624[accessed6 June 2014].

Lei et al., 2007 IPOG: A general strategy for T-Way software testing. [Online] available from: http://www.slideshare.net/Softwarecentral/ipog-a-general-strategy-for-tway-software-testing. [accessed 16 Apr 2014].

N.Borazjany, et al, 2012. Combinatorial Testing of ACTS: A Case Study,[online] availablefrom: http://www.google.com/url?sa [accessed October 2013].

Software testing, [online] Available from: http://en.wikipedia.org /wiki/ Software testing. http://www.developsense.com/pairwiseTesting.html, [accessed November 2013]).

Pan, 1999. Software testing.[online] Available from: http://users.ece.cmu.edu/~koopman / des_s99/sw_testing.[accessed May2014].

PictMaster. [online] Available from: http://en.sourceforge.jp/projects/pictmaster. [accessed 8 April 2014].

Practical Combinatorial Testing. [Online] Available from: csrc.nist.gov/groups/SNS/acts/documents/SP800-142-101006.pdf [accessed October 2013].