# A Case Study for Generating Test Cases from Use Cases

Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Jesús Torres, Arturo H. Centeno
*Department of Computer and Software Languages*
*University of Sevilla*
*Spain*
*{javierj, escalona, risotto, jtorres}@lsi.us.es*

***Abstract*— The verification of the correct implementation of use cases is a vital task in software development and quality assurance. Although there are many works describing how to generate test cases from use cases, there are very few case studies and empirical results of their application and effectiveness. This paper introduces a first case study that test the correct implementation of use cases in a web system and a command line system, analyses the results and exposes that generation of use cases has a successful about 80%.**

***Index Terms*—Test objective, Use case, Automatic generation, Testing tools, Case study, Empirical evaluation.**

## I. Introduction

Nowadays, use cases are a widely used technique to define the functional requirements of software systems. Several authors, like Cockburn [6], Ben Achour [4] or Escalona [9] [10], propose how to define use cases with UML Use Case Diagrams, which describes the relations between use cases and actor and between use cases and other use cases, in combination with templates for describing the behaviour of every use case and their preconditions, post-conditions, performance, priority, stability, etc. Templates are writing in narrative English with few or none formalism.

Two reports [7] [11] discovered two main gaps in the generation of test cases from use cases: lack of automatism and absence of empirical evaluation. The automatism of scenarios analysis written in natural language (first gap) has been resolved in our previous works [13] [14] using language patterns and regular expressions for extracting information from use case templates. This paper is focused in the second gap. Few empirical results about functional system testing have been published. So, the main goal of this paper and its original contribution is the execution of two cases studies to measure and evaluate the effectiveness of scenario analysis technique. The case study not only generates test cases, but implements and executes them to evaluate their effectiveness. The scenario analysis technique is a common technique for generating test cases from use cases. It identifies the scenarios from a use case and generates test cases from them. As mentioned before, the main contribution of our previous papers are the automatism of this technique.

This paper is organised as followed: section II describes the technique for generating test cases from use cases using use case scenarios. Then, section III describes systems under test and the preparation of the case study. Section IV describes the results of the case study. Section V introduces other related works. Finally, section VI exposes conclusions and ongoing works.

## II. An Overview of Scenario Analysis

As seen in section I, a use case is mainly defined by natural language and it is mainly composed of steps. In this paper, those steps are grouped in a main sequence, an alternative sequence and an erroneous sequence. An alternative sequence defines the steps that may be realised as an optional alternative to a step of the main sequence, while an erroneous sequence defines the steps that may be used if a step from the main sequence meets an error and is unable to exercise its behaviour. An example of use case, taken from the case study, is showed in table 2. The textual template is codified as a XML file to improve automation.

However, the natural language is often too ambiguous and generic to be automatically processed. Thus, the first task is to translate the behaviour of a use case into a more formal model. An UML Activity diagram has been chosen to define the behaviour of a use case. An Activity diagram allows indicating if an action is performed by the system or by an external action; it includes different execution flows; it does not need to expose information about the implementation of the system or its external interfaces. An example is shown in figure 1.

The algorithms used to extract information from a use case and to generate an activity diagram may be consulted in previous papers [13] and [15]. They have been implemented in an open-source software tool called TestGen (available in www.lsi.us.es/~javierj/). The result of this tool is shown in figure 1 according to the use case of table 1.

Table 1. Use case example

```
<useCase id="Search link by description">
  <description> A use case searches a set of links by their description and shows the results. </description>
  <mainSequence>
      <step id="1"> The visitor asks the system for searching links by description.    </step>
      <step id="2"> The system asks for the description. </step>
      <step id="3"> The visitor introduces de description. </step>
      <step id="4"> The system searches for links which match up with the description introduced by the visitor. </step>
      <step id="5"> The system shows the found results. </step>
 </mainSequence>
 <alternativeSteps>
    <astep id="3.1"> At any time, the visitor may cancel the search, then the use case ends.  </astep>
    <astep id="4.1"> If the visitor introduces an empty description, then the system
                     searches for all the stored links and step 5 is repeated.   </astep>
  </alternativeSteps>
  <errorSteps>
    <estep id="4.2"> If the system finds any error performing the search, then an error
                     message is shown and this use case ends.   </estep>
    <estep id="4.3"> If the result is empty, then the system shows a message and this use case ends.   </estep>
  </errorSteps>
</useCase>
```

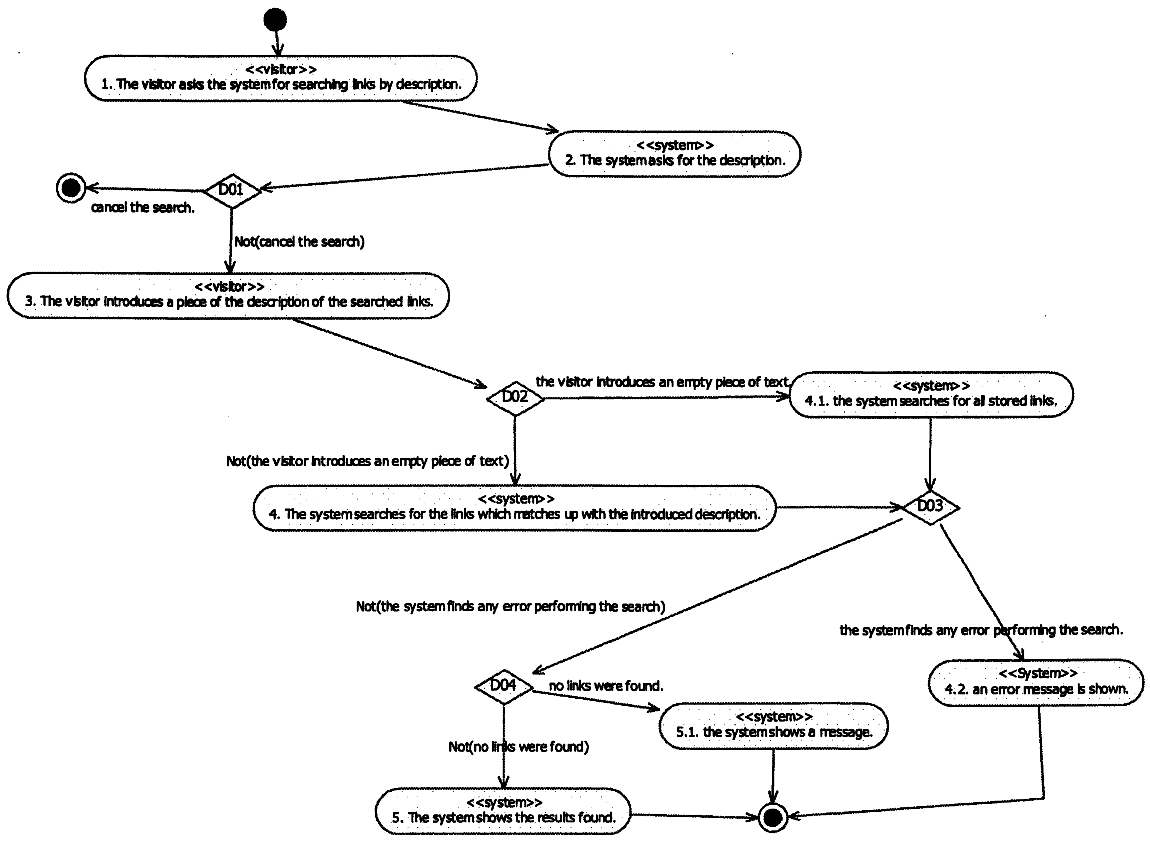
<<visitor>>
1. The visitor asks the system for searching links by description.
<<system>>
2. The system asks for the description.
D01
cancel the search.
Not(cancel the search)
<<visitor>>
3. The visitor introduces a piece of the description of the searched links.
D02
the visitor introduces an empty piece of text.
<<system>>
4.1. the system searches for all stored links.
Not(the visitor introduces an empty piece of text)
<<system>>
4. The system searches for the links which matches up with the introduced description.
D03
Not(the system finds any error performing the search)
the system finds any error performing the search.
<<System>>
4.2. an error message is shown.
D04
no links were found.
<<system>>
5.1. the system shows a message.
Not(no links were found)
<<system>>
5. The system shows the results found.

Fig 1. Activity diagram automatically generated.

Then, use case scenarios are derived from the activity diagram. The TestGen tool implements the all-nodes, all-transitions, and all-scenarios criteria to select scenarios. For the all-nodes criterion, the TestGen tool selects the paths that go across a higher number of actions until all the actions of the activity diagrams have been traversed at least once. For the all-transitions criterion, the TestGen tool selects the path that traverse a higher number of object-flow edges until all of them have been crossed at least once.

If the activity diagram has not got any loops, as in figure 1, the all-scenarios criterion selects the paths that go through all output object-flow edges from decision nodes at least once. If the activity diagram has got some loops, the all-scenarios criterion selects the paths that go through all output object-flow edges from decision nodes and all combinations among loops at least once. Table 2 shows an example of the paths and use case scenarios that have been obtained after applying the all-scenarios criterion in the activity diagram of figure 1 (each path is a test case). The numbers in each test case indicates the activities and decisions traversed from the activity diagram.

Table 2. Paths and a use case scenario.

```
Use case: Search link by description
The All-Scenarios Criterion
Test cases (Tc): 7

1: 1, 2, D01, End.
2: 1, 2, D01, 3, D02, 4.1, D03, 4.2, End.
3: 1, 2, D01, 3, D02, 4.1, D03, D04, 5.1, End.
4: 1, 2, D01, 3, D02, 4.1, D03, D04, 5, End.
5: 1, 2, D01, 3, D02, 4, D03, 4.2, End.
6: 1, 2, D01, 3, D02, 4, D03, D04, 5.1, End.
7: 1, 2, D01, 3, D02, 4, D03, D04, 5, End.
```

```
Use case scenario 1:

1: The visitor asks the system for searching
links by description.
2: The system asks for the description.
D01: The visitor cancels the search then the use
case ends.
End.
```

Second row in table 2 describes the steps performed by the test case number 1.

## III. CASE STUDY SETTINGS

The goal of this case study is to measure de effectiveness of the test cases generated from use cases using scenario analysis. For this reason, mutant systems, with different behaviour than the one described in its use cases are generated. Effectiveness is measured with the number of mutant killed (this means, different behaviour detected) by the test cases.

### *A. Systems under test*

Two systems have been tested in the case study: a CRUD web system (WEB) and a desktop system with a command-line interface (CML). Not all use cases have been tested. The statistics of the tested use cases of both systems are resumed in table 3.

The WEB system allows to maintenance an on-line link catalogue. The four use cases under test for WEB system are: add new link, search links by description (showed in table 1), show recent links and view details of a link.

Table 3. Use cases under test.

| | WEB | CML |
|---|---|---|
| *Use cases* | 4 | 3 |
| *Total number of steps* | 14 | 11 |
| *Total number of alternative steps* | 13 | 9 |

The CML system is a simple annotation application. The three use cases under test for CML system are: add a new note, erase all notes and list all notes. Implementation details of both systems are described in table 4.

Table 4. Implementation details.

| | WEB | CML |
|---|---|---|
| *Modules* | 20 | 1 |
| *Lines of code* | 4630 | 76 |

WEB system was developed in Java using Struts 2.0 framework. Modules include Java classes and JSP pages. CML was developed also in Java, as a stand-alone class, using standard packages only.

### *B. Mutating use cases*

A set of mutant versions of the two systems under test were codified. Those mutant versions were able to run successfully, but they exhibit a different behaviour than the one specified in their use cases.

Faults were introduced into systems using mutant operators. No classic code mutant operators were used, due our objective is not to change the source code (we do not test code). Our objective was mutant use cases for describing a different behaviour. So, the first step was to generate a new set of mutant operators for use cases. These mutant operators were obtained from the use cases fault model introduce by Binder [5]. The complete list of mutant operators used is showed in table 5.

Mutant operators were applied over use cases to generate mutant systems. A mutant system is a complete system with only one fault, which generates a different behaviour. Table 6 shows mutants obtained for each use case of the WEB system.

Table 7 shows mutants obtained for each use case of the CML system.

Even little use cases may have a big number of mutants and variations. An example of mutants obtained after apply the mutant operator 3 (sudden end of the use case) over the use case described in table 1 (Search link use case of the WEB system) is showed in table 8.

Table 5. Mutant operators for use cases.

| Id | Mutation operator |
|---|---|
| 1 | Logical operator from a condition of an alternative or erroneous step replacement. |
| 2 | Condition of an alternative or erroneous step always evaluated to true or false. |
| 3 | A sudden end step. |
| 4 | A step deletion. |
| 5 | A new step performed by the system addition. |
| 6 | Incorrect data admission or validation rules replacement. |
| 7 | Incorrect or incomplete information showed by the system. |
| 8 | An operation that may fail, always works correctly. |
| 9 | An operation that has not an erroneous step attached fails. |
| 10 | Information showed to the actor has fewer elements. |
| 11 | Step performer replacement. |

Table 6. Mutants for WEB system.

| *Use cases* | **Mutants obtained** |
|---|---|
| *Add new list* | 34 |
| *Search links* | 33 |
| *List recent links* | 15 |
| *View details of a link* | 10 |
| ***Avg. mutants per use case*** | 92 / 4 = 23 |

Table 7. Mutants for CML system.

| *Use cases* | **Mutants obtained** |
|---|---|
| *Delete all notes* | 15 |
| *Add new note* | 18 |
| *List notes* | 20 |
| ***Avg. mutants per use case*** | 53 / 3 = 17'7 |

Table 8. Examples of mutating a use case.

| **Mutants** |
|---|
| *Step 4:*<br>Instead of performing the query, the use case ends. |
| *Step 4.1*<br>If description is empty, then the use case ends. |

Figures 2 and 3 describe the number of mutants obtained from each mutant operator to the CML system (figure 2) and the WEB system (figure 3).

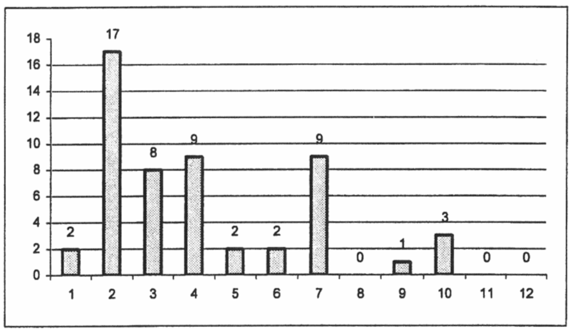

Fig. 2. Mutants obtained for each mutant operator (CML system).

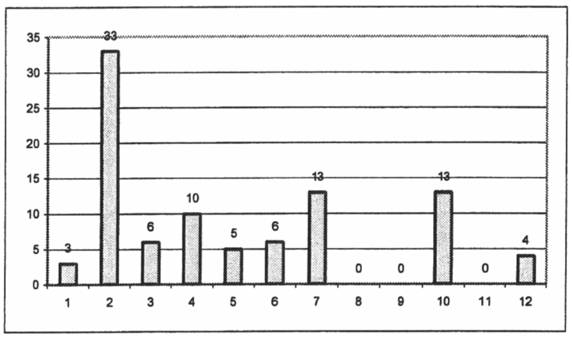

Fig. 3. Mutants obtained for each mutant operator (WEB system).

## IV. Case study results

### A. Test case generation

Test cases were generated (before obtaining mutant systems) using the supporting tool, TestGen, developed in the cited previous works.

Table 9 enumerates the test cases obtained after applying three coverage criteria (described in section II) over the activity diagram generated from each use case of the WEB system.

Table 9. Test cases for WEB use cases.

| *Use cases* | AllNodes | AllScenarios | AllTransitions |
|---|---|---|---|
| *Add new list* | 3 | 10 | 5 |
| *Search links* | 6 | 7 | 5 |
| *List recent links* | 3 | 3 | 3 |
| *View details of a link* | 1 | 3 | 3 |
| **Total:** | 13 | 23 | 16 |

Table 10 enumerates the test cases obtained after applying the three coverage criteria over the activity diagram generated from each use case of the CML system. In both tables (9 and 10), the littlest number of test cases is obtained with the all-nodes criterion and the bigger number is obtained with the all-scenarios criterion.

Table 10. Test cases for CML use cases.

| *Use cases* | AllNodes | AllScenarios | AllTransitions |
|---|---|---|---|
| *Delete all notes* | 2 | 4 | 4 |
| *Add new note* | 1 | 8 | 3 |
| *List notes* | 4 | 4 | 4 |
| **Total:** | 7 | 16 | 11 |

## *B. Test case codification and execution*

Taking every scenario as a test case, the number of scenarios calculated was 23 for WEB system and 16 for CML system. Then, test cases were codified with the help of two test harness described next. Test cases for all-nodes and all-transitions are subsets of the test cases for all-scenarios.

The test harness selected for the WEB system was JWebUnit (jwebunit.sourceforge.com). This tool interacts with the web system under test in the same way than a web browser. It also includes the JUnit tool for validating results.

For the CML system, an ad-hoc console test harness was developed. This test harness redirects the standard input and output. This one allows simulating a set of inputs from a user and evaluating the output of the system using the JUnit tool too.

After that, use case scenarios were codified in Java and executed over all the mutant versions generated in section III.B. Results for the test cases obtained with the all-scenarios criteria are listed in table 11.

Table 11. Results using all-scenarios criteria.

| | **WEB** | **CML** |
|---|---|---|
| *Killed mutants* | 76 | 45 |
| *Effectiveness* | 76 / 92 = 82'6% | 45 / 53 = 84'9% |

Results for the test cases obtained with the all-nodes criteria are listed in table 12.

Table 12. Results using all-nodes criteria.

| | **WEB** | **CML** |
|---|---|---|
| *Killed mutants* | 60 | 45 |
| *Effectiveness* | 60 / 92 = 65'2% | 45 / 53 = 84'9% |

Results for the test cases obtained with the all-transitions criteria are listed in table 13.

Table 13. Results using all-transitions criteria.

| | **WEB** | **CML** |
|---|---|---|
| *Killed mutants* | 72 | 45 |
| *Effectiveness* | 72 / 92 = 78'3% | 45 / 53 = 84'9% |

Results for the three criteria are the same in the CML system. This fact seems to indicate than it is possible to reduce the number of test cases from 16 to 7 by selecting the minimum number of test cases for each use case in table 9. The implementation of the all-nodes criterion tends to find the biggest path, this means, the path that traverse a bigger number of activities and decisions and, therefore, the test case that verify more steps. So, this fact suggested that size matters when testing use cases. This means that big paths, and test cases that exercise much of the steps of a use case, are better than little paths.

However, results from WEB systems invalidate these ideas. As showed in tables 12 and 13, a reduction of the number of test cases executed implies a reduction of the effectiveness (mutants detected), even when all steps of the use cases are exercised at least once (all-nodes) and all execution flows of the activities diagrams are traversed al least once (all-transitions). This fact is provoked for implementation details of the WEB system. As mentioned, this system has been codified using Struts framework. This tool imposes a concrete way of work, with object caching, session mechanisms, etc. So, if an operation where executed successfully the first time, second and other times the results are already available and operation is not repeated. This fact justifies the decrement of effectiveness when reducing the test case number and it suggests that a big number of use cases with the same test steps executed in different sequences are useful.

## V. RELATED WORKS

There are several papers and approaches about the testing of use cases defined in a textual tabular notation. An extensive list of references may be found in [7] and [11] reports. In next paragraphs, some of the most relevant approaches are summarised.

TDE/UML approach, [8], expresses a use case as a UML activity diagram and uses the Category-Partition method [12] to generate test cases. However, the approach does not indicate if the activity diagram may be generated automatically from the use cases, nor the format in which the use cases must be defined.

TOTEM [18], Requirement-based Contract [17] and the CowSuite [2] approaches expressed a use case as an UML sequence diagram. The sequences of messages are expressed as regular expressions and are combined between them to generate test cases. We found some problems using sequence diagrams. It is very difficult to express alternative or erroneous sequences in the same diagram. Information about architecture and internal implementation, like classes and messages are also needed, so it cannot be applied in the early phases of the development.

Other approaches work directly with natural language, like references [16] and [19]. All of them propose a simple combinational explosion among all scenarios in a use case. These approaches are quite simple and omit many important aspects, like coverage, test values, expected results or test implementation.

As mentioned in introduction, there are very few works that describes case studies. One of them is [3]. In this paper describes the testing of a real E-Ticket system for the Netherlands. For testing, a test scenario was generated (by hand) from each use case scenario, then, each test scenario was instantiate into a test case with concrete test values. Results were satisfactory, discovering and resolving all critical issues and about 50% of medium ones.

## VI. CONCLUSIONS

It is not the same mutating test cases for generating mutant systems than testing real system with real faults. Some studies, like [1] exposes that real system and faults are less elaborated than mutant faults. However, the results described in this paper indicates that it may be valuable apply the use case scenario analysis to real systems. Results from section IV.B suggested that it is not only important to exercise all steps from a use case but exercise them in different combinations. Each step is codified into test code once and, then, used in every test case that exercises the step, so it is easily to generate a big amount of test cases with little codification effort. Moreover, results in section IV.B also exposes that the automatic generation of software testing may detect a valuable amount of errors. However, case studies and experiments are also hard to perform due the generation and implementation of mutants is not automatic. The biggest amount of time dedicated to this case study have been spent in developed mutant systems and executing test cases over mutant systems, instead of generating of codifying test cases.

The main ongoing work is to repeat this case study with the operational variable analysis technique [5] and compare results.

## VII. ACKNOWLEDGES

This work is supported by the Ministry of Science and Education under the National Program for Researching, Development and Innovation, project QSimTec (TIN2007-67843-C06-03) and REPRIS (TIN2005-24792-E).

## REFERENCES

[1] Briand L.C. Labiche Y. 2005. Is Mutation an Appropiate Tool for Testing Experiments?. *International Conference of Software Engineering ICSE'05*. St. Louis, Missouri, EEUU.

[2] Basanieri F. Bertolino A. Marchetti E. 2002. The Cow_Suite Approach to Planning and Deriving Test Suites in UML Projects. *Lecture Notes In Computer Science* 2460 pp. 383-397.

[3] Roubtsov S. Heck P. 2006. Use Case-Based Acceptance Testing of a Large Industrial System: Approach and Experience Report. *TAIC-PART 06*. Windsor, UK.

[4] Ben Achour C. 1998. Writing and Correcting Textual Scenarios for System Design. *Natural Language and Information Systems Workshop*. Vienna, Austria.

[5] Binder R. V. 2000. *Testing Object-Oriented Systems. Addison-Wesley*. USA.

[6] Cockburn, A. 2000. *Writing Effective Use Cases*. Addison-Wesley 1st edition. USA.

[7] Denger, C. Medina M. 2003. Test Case Derived from Requirement Specifications. *Fraunhofer IESE Report*. Germany.

[8] Ruder A. 2004. UML-based Test Generation and Execution. *Rückblick Meeting*. Berlin. Germany.

[9] Escalona M.J. 2004. *Models and Techniques for the Specification and Analysis of Navigation in Software Systems*. Ph. European Thesis. Department of Computer Language and Systems. University of Seville. Seville, Spain.

[10] Escalona M.J. Gutiérrez J.J. Villadiego D. León A. Torres A.H. 2006. Practical Experiences in Web Engineering. *15th International Conference On Information Systems Development*. Budapest, Hungary, 31 August – 2 September

[11] Gutiérrez, J.J., Escalona M.J., Mejías M., Torres, J. 2004. Comparative Analysis of Methodological Proposes to Systematic Generation of System Test Cases. 3° Workshop on System Testing and Validation. Paris. France.

[12] Ostrand T. J., Balcer M. J. 1988. Category-Partition Method. *Communications of the ACM*. 676-686.

[13] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. Derivation of test objectives automatically. *Fifteenth International Conference On Information Systems Development (ISD06)*. Budapest, Hungary, 31 August – 2 September, 2006

[14] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. Towards a Complete Approach to Generate System Test Cases. *ICEIS Doctoral Consortium*. Oaphos, Cyprus.

[15] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. Modelos Y Algoritmos Para La Generación De Objetivos De Prueba. *Jornadas sobre Ingeniería del Software y Bases de Datos JISBD*. Sitges. Spain.

[16] Heumann , J. 2002. Generating Test Cases from Use Cases. *Journal of Software Testing Professionals*. EEUU.

[17] Nebut C. Fleury F. Le Traon Y. Jézéquel J. M. 2006. Automatic Test Generation: A Use Case Driven Approach. *IEEE Transactions on Software Engineering* Vol. 32. 3. March.

[18] Labiche Y., Briand, L.C. 2002. A UML-Based Approach to System Testing, *Journal of Software and Systems Modelling (SoSyM)* Vol. 1 No.1 pp. 10-42.

[19] Naresh, A. 2002. Testing From Use Cases Using Path Analysis Technique. *International Conference On Software Testing Analysis & Review*. EEUU