# A FORMAL SOFTWARE TESTING TECHNIQUE

M. M. Baig and A. A. Khan,

Corresponding author email: mbaig2000@gmail.com, dransark@yahoo.com

**ABSTRACT:** Software Testing is defined as a single phase activity in software development process by using water fall model. We have been working on a research project on formal software testing technique. In this paper, contrary to usual practice, a built-in testing module at each stage of software development process is proposed. The problem description is first expressed in a formal notation to get the problem-definition, in an unambiguous and formal format. This formal description of the problem will be more accurate and will form a formal test bed in the tabular form. This test bed will facilitate different types of relevant testing at each phase of system development process, both vertical and horizontal testing. The said software testing technique consists of three modules. This paper covers the work done in the first module.

**Key words:** Formal notation, Software Testing, Specification, Verification and Validation.

## INTRODUCTION

Typical software testing costs vary based on the criticality of the software development project from 40% to 85% of the entire system development process as per (Pushkar, 2004; Gaudel, 1988). To remedy this over expenditure of money and time, it is proposed that a formal method for testing be incorporated in the software development methodology. One weak point in software development process is that testing is undertaken as a single step at the end of software development process as described in (Kaner, et al., 2001; Ludwig, 2003; Burnstein, 2003; Young, 2005; Last, 2004; Loveland, etal., 2004) and the second problem is that organizations pressurize to release the product before it is completely tested. As described in (Tian, 2005; Pushkar, 2004; Hoglund 2004) as special notes of considerations and challenges that are particular to testing are: improve testing process, define requirements formally, prove the concept, champion product testability, design for sustainability, and plan for deployment, face challenges of success. In (Dalal, 2004) the testing issues defined are: no formal methodology for system development, no formal walkthroughs, no documentation, absence of interactions between groups, no stress testing. The paper also provides summary of testing problems and emphasizes that "testing is so important that it should be considered from project inception".

## MATERIALS AND METHODS

The proposed technique consists of three modules:

1- A Formal Notation to describe specifications;
2- A Formal Test bed (in a matrix form);
3- A set of algorithms and tools.

**Formal Notation to Describe Specifications (FRS)**

A software testing system is defined as a vector = [boundary specifications, generic object type],
Where
Boundary specifications = [output attribute vector, input attribute vector]
Outputs = [output attribute vector]
Inputs = [input attribute vector]
Generic object type:
Object name
[object type identifier]
[Domain{attributes}]
[{Sub domains{attribute}]
{Probability}]
[identified by{key attributes}]
[triggered by{output stimulus attributes}]
[function of{input attributes}]
[derivation is{computing expression}]
[members {attributes}]
[Synonym {synonym title}]
[comment {comments if necessary}]
End.
Above generic objects or structures are optional and may be used as per requirement of the tester/developer
Where;
Object type identifier= input/output/relation/domain/function/item
Identifier = Key attribute/(s) which uniquely identifies one snap shot (or an occurrence) of generic object type.
Derivation expression = computation procedure of a derivation function.
Output trigger = input attribute name/ET/if condition.
Option = qualified option/unqualified option.
Qualified option = option on "generic attribute name".
Unqualified option = option either {attributes} Or {attributes}

Test Bed = is an environment that contains all details, software, hardware needed to test a software component or software system.

**Example System**: On the last working day of every month, a pay slip is generated by MIS department which is distributed to all employees. On first day of every month salaries of all employees are credited in the bank accounts of all employees. This pay slip provides all details of the salary i.e. basic pay, allowances and deductions. Net salary = gross salary - deductions, where gross salary = basic pay salary + allowances. The allowances are: house rent, conveyance, computer, entertainment, medical, ad-hoc relief, senior post, Chairman etc. The deductions are: income tax, BF, PF, Group insurance, teacher society contribution and advances taken.

**The problem described in formal notation (module 1) is as follows:**

**System title is: Employees monthly salary pay slip**

**Boundary Specification vectors**

Output vector Monthly pay slip;
Input vector employee detail, income details, deductions, Advances & loans;
Comment (every month a pay slip is generated for each Employee.)
End.

**Monthly Pay Slip**

Output vector
Trigger ET;
Identifiers key ( e #, name );
Attributes (employee#, name, designation, grade, status, bank acc#, department)
Cardinality ( 0 . . . 1000);
Extension 1000;
Frequency I per month
End.

**Employee details**

Input vector
Attributes ( employee#, name, designation, grade, status of post, bank acc#, department.);
Cardinality ( 0 . . . 1000);
Extension 1000;
Frequency one per month;
Comment ( new employee details type through keyboard & updates done every month.)
End.

**Income details**

Input vector
Attributes (e#, basic pay, computer allowance, conveyance allowance, medical allowance, house rent)
Cardinality ( 0 . . . 1000);
Extension 1000;
Frequency one per month;
End.

**Deductions**

Input vector
Attributes (e#, income text, GF, GPF, GI, car advance, house advance);
Cardinality ( 0 . . . 1000);
Extension 1000;
Frequency one per month;
End.

**Advances & loans**

Input vector
Attributes ( car purchase, house purchase, computer Purchase )
Cardinality ( 0 . . . 1000);
Extension 1000;
Frequency one per month;
End.

**Employee number**

Item
Domain integer ( 1…1000);
Status constant;
Comment ( status stands for whether item is constant Or requires derivative )

**Employee name**

Item
Domain string;
Status constant
End.

**BPS number**

Item
Domain integer( 1…22);
Status constant;
End.

**Basic pay**

Item
Domain real(0…100000);
Status constant;
End.

**Income tax**

Item
Attributes gross pay, deductions, IT rates;

Domain real(0….50,000);
Identifiers e#;
Status derived
Filter option 1;
Trigger govt. notice;
Comment if item is derived than derivation is described
End.

**Option 1**

Filter if income >=60,000 than select
End.

**Derivation**

If income > 60,000 and < 100000
IT rate = 10%;
Else if income > 100000 and < 200000,
Then IT rate 15%;
Else if income > 200000 and < 300000,
Then IT rate 20%;
Else in come > 300000
Then IT rate 25%;
End.

| | |
|---|---|
| **Net salary** | |
| | Item<br>Identifiers e#;<br>Domain real(0…00000);<br>Attributes gross pay, deductions, advances;<br>Status derived;<br>Filter option 2;<br>Trigger pay slip<br>Comment all regular employees are subject to allowance, deduction etc. but contract employees salary is fixed.<br>End. |
| **Option 2** | |
| | Filter if employee status = regular than select |
| **Derivation** | |
| | If employee = regular than net salary = gross pay – ( deduction + advances )<br>End. |

## RESULTS AND DISCUSSION

It is expected that after completion of the entire three modules we will be able to provide enormous assistance to software-tester. The technique will provide a complete software testing technique which is expected to be: accurate, patent, structured and unambiguous technique to test software at each step of software development process contrary to existing practice i.e. testing at a single step only.

The FRS provides a flexible and unambiguous (easy to use) capability to all kind of software developers.

**Conclusions:** This paper describes the development of a Software research project consisting of three modules: Formal notation for Requirement Specification (FRS); a test bed; a set of testing tools. On the basis of the state of art literature survey so far, and working on 1st module of my research project and then applying FRS (i.e. 1st module) on a small typical example system the robust expressive quality of the FRS has been shown. It is expected that after completion of the entire project we will be able to provide great assistance to software tester a complete software testing technique which will have: precise, clear, structured and unambiguous capability of describing the problem in FRS and then automatically transforming the problem in FRS in a tabular form (Test Bed) from where we will easily take up validation and verification of the functions described in formal specification.

**Acknowledgment:** The authors acknowledge all kind of support by NED University of Engineering & Technology and specially the Vice Chancellor as chairman of Advanced Studies & Research Board.

## REFERENCES

Burnstein, I., Practical Software Testing 1st Ed. Springer, (2003).

Kaner, C., J. Bach and Pettichord, B., Lesson Learned in Software Testing. Wiley, Ist Ed. (2001).

Dalal, G., Software Testing Fundamentals Methods and Metrics. American Society for Quality. (2004).

Gaudel, C., Algebraic Specifications and Software Testing: Theory and Application. University Paris Sud. (1988).

Hoglund, G., Exploiting Software: How to break code. McGraw. (2004).

Ludwig, E., Software Testing should reveal errors not avoid them. Washington information source. (2003).

Last, M., Artificial Intelligence Methods in Software Testing. World Scientific Publishing Company. (2004).

Loveland, S., G. Miller, R. Prewitt Jr. and M. Shannon, Software Testing Techniques finding the defects that matter. Charles River Media. (2004).

Puskar, H., Effective Software Testing, 50 specific ways to improve your Testing. American Society for Quality, 6. (2004).

Tian, J., Software Quality Engineering. Wiley-inter Sc. (2005).

Young, M., Software Testing and Analysis: Process, Principles, & techniques. John Wiley & Sons. (2005).