

Generation of Test Cases from SRS – Proposed Model for Testing GUI Screens

SUMITA

Student, M.Tech (IT)

USIT, GGSIPU

sumeeta@gmail.com

Guru Gobind Singh Indraprastha University, Kashmiri Gate, DELHI – 110403. URL :www.ipu.ac.in

Abstract— Software robustness is a problem that everybody cares about but requires a lot of planning, time and implementation strategies. Typically a project is assigned several weeks devoted to testing, mostly in the weeks before deployment. Of course, most software ends up behind schedule and over budget, and testing is the first thing to get reduced or cut. Thus, much commercial software gets only a couple of days of testing before it is shipped, which may possibly dent its quality hugely.

Testing is as important as the development is. So there have been a number of efforts to start testing as soon the development commences. This not only prevents a bug from piling on from specifications phase to post-development testing and fixing phase but also reduces cost of fixing the bug.

This paper describes a model for preparing test cases from SRS document for GUI screens given in SRS. It will suggest ways to write test cases from the information available in SRS.

We propose a theoretical model to represent an interface screen given in SRS and then devising test cases from that.

Case study tries to give example of theoretical model of an interface given in SRS and derived test cases from it.

I. INTRODUCTION

Software testing is essential part of software development life cycle that aims at one of the two things: to make a judgment about quality or acceptability and to discover problems [1].

Software development life cycle formally begins with the preparation/production of the SRS (Software Requirement and Specifications) document. SRS formally lays down the specifications for a particular software product, program or a set of programs, based on which the development progresses. SRS also serves as the base document for validation purposes. Thus, SRS can also serve as a ground for generating test cases prior to any further activity.

Each specification in SRS may lead to a test case, thereby making a test suite available prior to any designing. This can be exploited for test-driven designing and coding.

II. SRS AS A GROUND FOR TEST CASE GENERATION

An SRS includes, apart from other things, the following, which may serve as a ground for generating test cases: *use cases, external interfaces, software product features, validity checks and sequencing information.*

Since there is no way to determine the internal structures of the system components from SRS, test cases pertaining to the definition of Black Box testing can be created at this stage.

There has been some earlier work for generating test cases from specifications. Use cases can serve as a ground for generating test cases. [3,4,5]. In one of these approaches, scenarios are generated from use cases wherefrom a behavioral model for the use case scenario is derived. It is then used to determine the test case categories and test cases.

The traditional approaches for generating test cases for black box testing (such as boundary value analysis, robustness testing, worst case testing, cause effect graphing technique, equivalence class testing, decision table based testing technique etc. [2]) can be well applied on the specifications available in the SRS considering the software product features and validity checks.

None of these techniques directly cater to testing the GUI screens available in SRS. Thus, here I propose a theoretical model to devise test cases for GUI screens given in SRS.

III. PROPOSED THEORETICAL MODEL FOR GENERATING TEST CASES FOR GUI SCREENS GIVEN IN SRS

In this model, each GUI screen is represented theoretically using a proposed pseudo language for GUI representation (PLGR). Each GUI screen would be represented as *Interface* that may contain GUI objects such as *text boxes, check boxes, option buttons, list boxes, combo boxes, labels* etc. Each *Interface* may invoke components like *dialog windows*.

Proposed definitions for this model are:

Interface: The GUI screen user interacts with.

Attributes: components contained in the Interface or invoked by the interface. Attributes can be:

NAttributes (Non-Participating attributes) are the ones that don't contribute to output in any way such as some informative labels.

PAttributes (Participating attributes) are the ones that contribute to output in some way such as text boxes accepting input values, check boxes etc. The participating attributes (PAttributes) are further categorized as:

IN – *Input* type Attributes that receive input.

OU – *Output* Type Attributes that display/provide output.

IO – *Input Output* Type Attributes.

AC – *Action* type Attributes that invoke some action i.e. cause some event to occur.

Events are the events that take place when the Interface is active.

Flow Sequence lists the possible sequences of attributes' combinations and events generating some output or causing some change or invoking other events.

The definition of *Interface* contains detailed definitions of each of its attributes, events and flow sequences. The template for defining Interface is:

```

INTERFACE <Interface-Name> IS {
  PURPOSE:: <purpose>;
  NPATTRIBUTES::
    NPAttribute-name1: <attribute type1> {
      IVALUE : <initial value>
    }; ... /* other NPAttributes' definitions*/
  PATTRIBUTES::
    PAttribute-name1: <attribute type> IN {
      IVALUE : <initial value>,
      RANGE : <lower value> to <higher value>,
      LIST : { <value1>, <value2>, ... },
      RULE1: VALUE <relational op.> {comparison-value},
      ... /* more rules*/
    }; ... /* other PAttributes' of IN type */
    PAttribute-name2: <attribute type> AC {
      IVALUE : <initial value>,
      LIST : { <value1>, <value2>, ... },
      EVENTS : <event1>, <event2>
      ... /* more events*/
    }; ... /* other PAttributes' of AC type */
    PAttribute-name3: <attribute type> OU {
      OVALUE: {<possible output messages>} |COMPUTED,
      RESULTOF : {<event1>, ...},
      CONTROL_PASSEDTO : <interface>.<attribute>
    }; ... /* other PAttributes' of AC type */
  EVENTS::
    Event-name1 {
      NORMAL OUTCOME: <attr list of OU type>,
      ALTERNATIVE OUTCOME: <attr. list of OU type>,
      ... /* more alternative outcomes */
    }; ... /* more events */
  FLOW SEQUENCES::
    FS1 : { <attr.list>,<event list>;
    ... /* more flow sequences */
};

```

Fig 1. Template of PRGL Interface

IV. CASE STUDY

I present a case study to represent a GUI screen in proposed PLGR. The sample screen is a simple login screen. As per proposed PRGL, the theoretical model of this sample GUI screen will be:

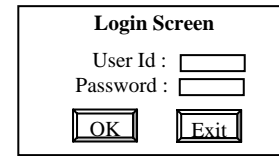


Fig. 2. Sample GUI Login Screen.

```

INTERFACE Login IS {
  PURPOSE:: "To validate userid & password;" ;
  NPATTRIBUTES::
    LblHeading: Label {
      IVALUE : "Login Screen"
    }; ... /* lblUserId & lblPwd's definitions*/
  PATTRIBUTES::
    txtUserId: TextBox IN {
      IVALUE : NULL ,
      RANGE : 000001 to 999999,
      LIST : { },
      RULE1: VALUE <> {NULL},
      RULE2: VALUE EXISTS IN{"Users.txt"}
    };
    txtPwd: TextBox IN {
      IVALUE : NULL ,
      RANGE : ,
      LIST : { },
      RULE1: Len(Value)=6 ,
      RULE2: VALUE <> {NULL},
      RULE3: VALUE EXISTS2 IN{"Users.txt"},
    };
    cmdOK: CommandButton AC {
      IVALUE : "OK" ,
      LIST : { },
      EVENTS : Click
    }; ... /* definition of cmdExit */
    DialogOK: Dialog OU {
      OVALUE : {"Login OK. Access Granted"},
      RESULTOF : {cmdOK_Click},
      CONTROL_PASSEDTO : NextInterface.Attr1
    };
    DialogError: Dialog OU {
      OVALUE : {"Invalid UserID", "Invalid Password",
        "UserId or Password blank" }
      RESULTOF : {cmdOK_Click},
      CONTROL_PASSEDTO : Login.txtUserId
    }; ... /* definition of DialogExit */
  EVENTS::
    cmdOK_Click {
      NORMAL OUTCOME: DialogOK
      ALTERNATIVE OUTCOME: DialogError
    }; ... /* defn of cmdExit_Click event*/
  FLOW SEQUENCES::
    FS1 : { txtUserId, txtPwd, cmdOK_Click};
    FS2 : { txtUserId, cmdOK_Click};
    FS3 : { txtPwd, cmdOK_Click};
    FS4 : { cmdOK_Click};
    /* Flow sequences containing cmdExit_Click event*/
};

```

Fig 3. Login Interface in proposed PRGL

Once the PLGR model for interface is ready, test cases

¹ Possible attribute types are : TextBox, ListBox, ComboBox, CheckBox, OptionButton, Label, CommandButton, Dialog etc.

² Assuming that EXISTS also check for validity of password whether it matches corresponding user id.

pertaining to input domain and output domains can be created as per following pseudo logic. There must be at least one test case for each flow sequence of the interface.

A. Devising test cases for Input Domain

1. For each Flow Sequence in interface
2. $TS^3 = attr_n = \{\emptyset\}$
3. If attribute is IN or IO type
4. If attribute has a RANGE defined
5. $TS_n = TS_n \cup \{BVA^4 \text{ test cases based on range}\}$
6. $TS_n = TS_n \cup \{\text{Robustness test cases based on range}\}$
7. $TS_n = TS_n \cup \{\text{Worst case test cases based on range}\}$
8. If attribute has a LIST defined
9. $TS_n = TS_n \cup \{\text{Equivalence Class Test Cases}\}$
10. If attribute has RULEs defined
11. $TS_n = TS_n \cup \{\text{Test cases pertaining to each rule}\}$
12. End
13. $TS = TS_1 \cup TS_2 \cup \dots \cup TS_n$

B. Devising test cases for Output Domain

1. For each Flow Sequence in interface
2. For each Event
3. $TS_{on} = TS_{on} \cup \{\text{test cases for normal outcome}\}$
4. $TS_{on} = TS_{on} \cup \{\text{test cases for alternative outcome}\}$
5. End
6. End
7. $TS = TS \cup TS_{o1} \cup TS_{o2} \cup \dots \cup TS_{on}$

For the sample GUI Login screen, based on above logic, the generated Test Suite will be as follows (For simplicity sake, I am not considering Robustness and Worst case test cases.):

Input Domain Test Cases

For FS1 in interface Login

$TS_1 = txtUserID = \{\emptyset\}$

txtUserld is IN type

It has RANGE Defined

$TS_1 = TS_1 \cup \{000001, 000002, 50000, 999998, 999999\}$

It has Rules defined

$TS_1 = TS_1 \cup \{\text{Null, Non-Null}\} \quad /* \text{txtUserID.Rule1}*/$

$TS_1 = TS_1 \cup \{\text{Value existing in File, Value not existing in file}\} \quad /* \text{txtUserID.Rule2}*/$

TxtPwd is IN type

$TS_2 = txtPwd = \{\emptyset\}$

It has Rules defined

$TS_2 = TS_2 \cup \{\text{pwd with 6 chars, pwd} < \text{6chars, pwd} > \text{6chars}\} \quad /* \text{txtPwd.Rule1}*/$

$TS_2 = TS_2 \cup \{\text{Null, Non-Null}\} \quad /* \text{txtPwd.Rule2}*/$

$TS_2 = TS_2 \cup \{\text{Value existing in File, Value not existing in file}\} \quad /* \text{txtPwd.Rule3}*/$

$TS = TS_1 \cup TS_2$

... repeat for other flow sequences.

Output Domain Test Cases

For FS1 in interface Login

$TS_{o1} = \{\emptyset\}$

For event cmdOK_Click

For Normal Outcome

$TS_{o1} = TS_{o1} \cup \{\{\text{txtUserID exists in File}\}, \{\text{txtPwd exists in File}\}\}$

For Alternative Outcome

$TS_{o2} = TS_{o2} \cup \{\{\text{txtUserID doesn't exist in File}\}, \{\text{txtPwd doesn't exist in File}\}\{\text{txtUserID is NULL}\}, \{\text{txtPwd is NULL}\}\}$

$TS = TS \cup TS_{o1} \cup TS_{o2}$

V. RESULT ANALYSIS

With the above-proposed theoretical model, I could prepare a test suite containing at least a test case to meet each possible input condition, output condition and event outcomes in normal and alternative cases.

VI. CONCLUSION AND FUTURE WORK

The theoretical model I suggested in this paper is purely based on specifications available in SRS. Since at the stage of SRS, no structural information is available, only black box testing is possible. For testing GUI screens, no direct method was available for which this model has been suggested. The completion of this work will result in to complete development of pseudo language for GUI representation (PLGR) and thereby complete development of pseudo logic for developing test cases from PLGR model.

In future, this work may further be enhanced by the development of a compiler for PLGR and by creating a testing tool out of it.

³ Test Suite

⁴ Boundary Value Analysis

ACKNOWLEDGMENT

The work described in this paper was conducted under the guidance of Dr. Yogesh Singh, Professor, USIT, Guru Gobind Singh Indraprastha University, Delhi –110403 (India).

REFERENCES

- [1] Paul C. Jorgensen, “Software Testing – A Craftsman’s Approach,” 2nd ed. CRC Press, 2002, pp. 3-4.
- [2] Glenford J. Myers, “The Art of Software Testing”, 2nd ed., John Wiley & Sons, Inc., 2004
- [3] C. Nebut, F. Fleurey, J-M. Jézéquel and Y. Le Traon, “Automatic Test Generation: A Use Case-Driven Approach”, IEEE Transactions on Software Engineering, , 32(3):140, March 2006.
- [4] Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Jesús Torres, “An Approach to Generate Test Cases from Use Cases”, University of Seville, Spain, 2006
- [5] Dave Wood & Jim Reis, “Use Case Derived Test Cases,”. STAREAST on Software Quality Engineering Conference., 1999