COMBINATORIAL TEST TOOL
DESIGN DESCRIPTION

DVA313

Group - 2

3rd December 2015

Contents

1. Introduction.. 1
2. High level description of the functionality 1
3. System overview 2
4. Software Architecture 2
5. Detailed software design 3
6. GUI design 5

1. Introduction

Most software failures are induced by a single factor faults or combination of factor faults. Combinatorial testing tool helps in detection of failures that can arise due to certain combination of inputs.

Some benefits of having a combinatorial testing tool are:

- Improved quality of the product
- Defects are found at an earlier stage, when it is cheaper to fix them
- Makes testing of the system more efficient

Our project is to develop this combinatorial test tool for Bombardier's Train Control Management System (TCMS). The train management system handles most of the train's operations like heating, opening and closing the door, the braking system and the engine.

The test tool should be written in C# language, running on Windows and also provide a GUI. User imports an XML file into the GUI and choose either random or base choice technique to generate different input combinations. The generated test cases will be saved by the user into a CSV file which will be used for testing TCMS.

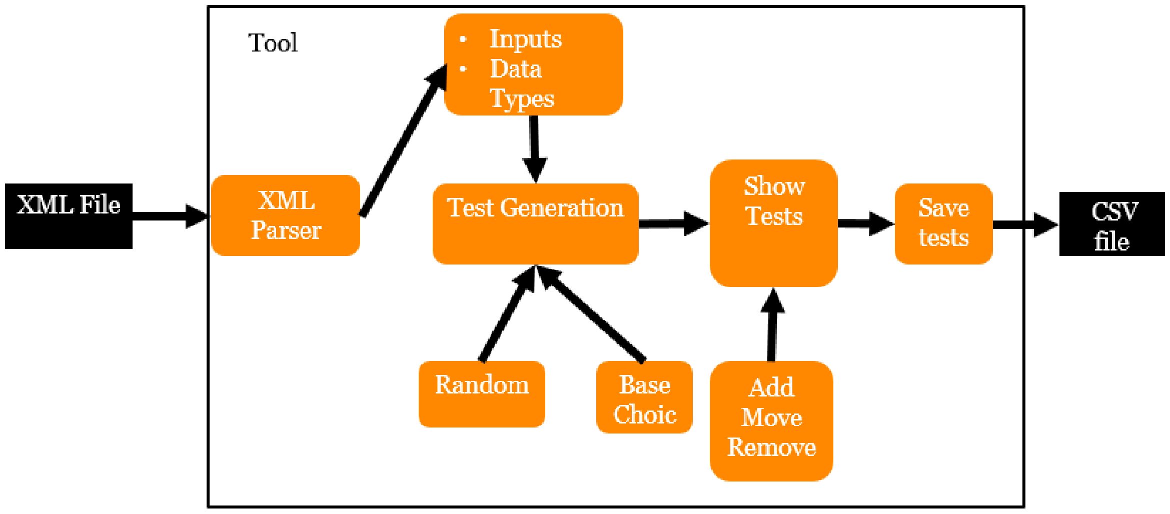


Figure 1. Basic functionality diagram

2. High level description of the functionality

The combinatorial test tool should be able to generate test cases based on the variables and data types present in the XML file which serves as input to the tool. The tool should allow users to choose whether the test cases should be generated based on the random technique or the base choice technique. If random technique is selected then users should be able to enter the number of test cases as well as the interval of each input value. The interval can be single or multiple. If base choice is selected then the users will input a base value and an interval for each input. Based on the selections, test cases should be generated. Users should be able to save the test cases into a CSV file which will be used for testing the TCMS software. The tool may also allow users to add, remove and reorder test cases. Optionally, the program should be able to run as a DLL callable by other applications, allowing the application to set the input values instead of a human user.

3. System erview

Inside Bombardier's trains they have a software system called TCMS. This system controls operations such as braking, heat level of engines, and opening and closing of doors etc. Since TCMS is controlling such crucial parts of their trains, it is of high concern to test it extensively. Currently this is done by technicians that manually plot test values into a test program.

The purpose of our program is to reduce the workload on the technicians and at the same time provide the possibility to run more tests on TCMS. The computer generated XML-files that holds the names and datatypes of each variable serves as the input to the test tool. And the CSV file generated from the tool will be used as input for testing the software of TCMS.

By doing this they can generate many more tests and with random values that would take too long for the technicians to manually plot into the test program. With a higher number of tests run the chance of finding critical bugs in TCMS increases.

4. Software Architecture

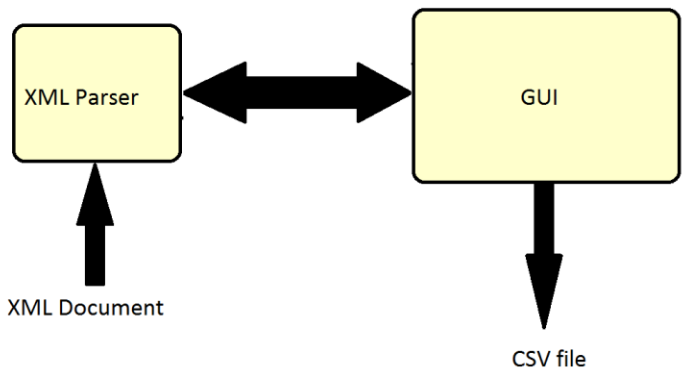


Figure 2. Software Architecture

The architecture of our Combinatorial Test Tool consists of two main independent entities: the XML Parser and the GUI. The Parser retrieves a document and sends the relevant data from it to the GUI where it is presented. From the GUI, the user can then generate tests, modify the values of the given variables, add interval ranges and save the generated tests to a CSV file. More detailed descriptions will be shown in the “Detailed Software Design” section.

5. Detailed software design

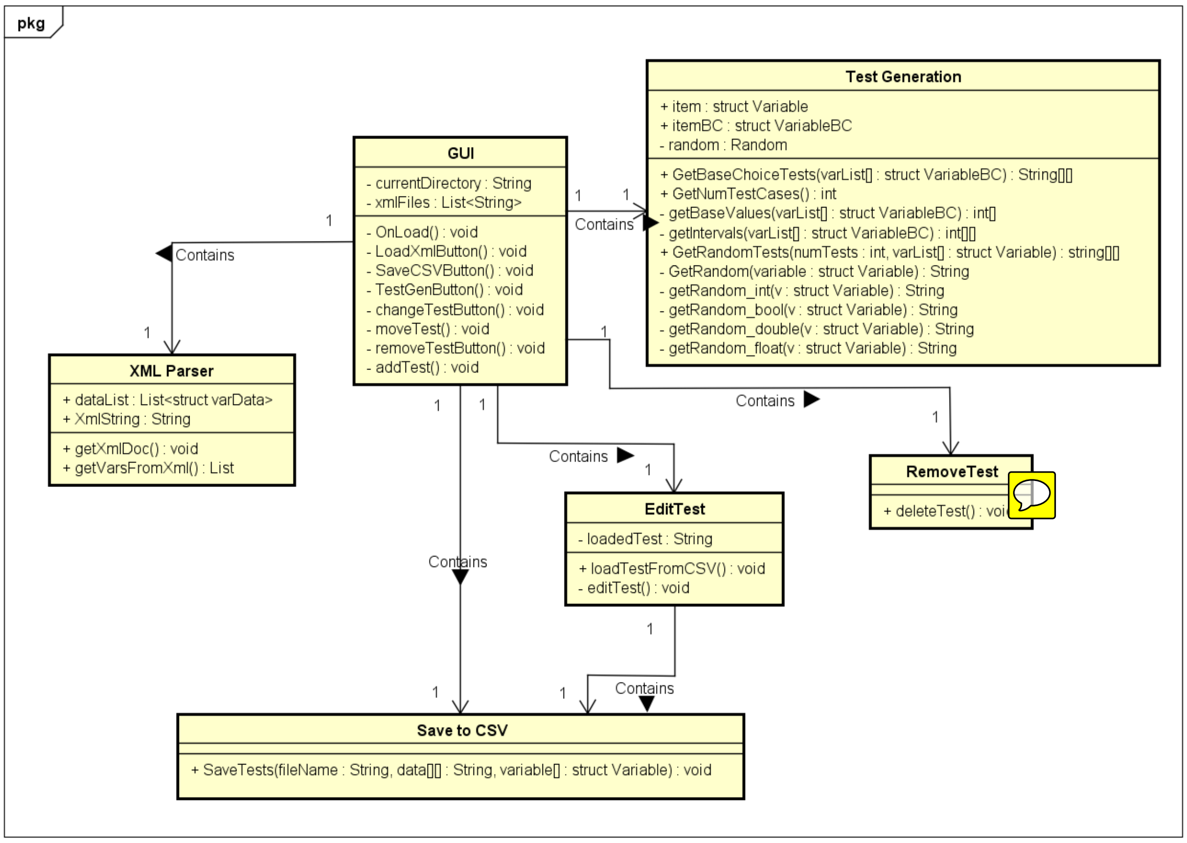

pkg
Test Generation
+ item : struct Variable
+ itemBC : struct VariableBC
- random : Random
+ GetBaseChoiceTests(varList[] : struct VariableBC) : String[][]
+ GetNumTestCases() : int
- getBaseValues(varList[] : struct VariableBC) : int[]
- getIntervals(varList[] : struct VariableBC) : int[][]
+ GetRandomTests(numTests : int, varList[] : struct Variable) : string[][]
- GetRandom(variable : struct Variable) : String
- getRandom_int(v : struct Variable) : String
- getRandom_bool(v : struct Variable) : String
- getRandom_double(v : struct Variable) : String
- getRandom_float(v : struct Variable) : String
GUI
- currentDirectory : String
- xmlFiles : List<String>
- OnLoad() : void
- LoadXmlButton() : void
- SaveCSVButton() : void
- TestGenButton() : void
- changeTestButton() : void
- moveTest() : void
- removeTestButton() : void
- addTest() : void
XML Parser
+ dataList : List<struct varData>
+ XmlString : String
+ getXmlDoc() : void
+ getVarsFromXml() : List
RemoveTest
+ deleteTest() : voi
EditTest
- loadedTest : String
+ loadTestFromCSV() : void
- editTest() : void
Save to CSV
+ SaveTests(fileName : String, data[][] : String, variable[] : struct Variable) : void
Contains
1
1

Figure 3. Class diagram of the tool

The software consists of a GUI, XML Parser, Test Generation and a Save function. The GUI handle all the graphical aspects and forward the data between the different entities. It presents the different Data Inputs gotten from the XML Parser that should be changed and used in the test generation. The Test Generation provides the GUI with data to be shown to the user.

The XML Parser gets the XML document from a directory provided by the GUI. It then extracts the input variables from the document and saves them in a list of structs named varData (that contains the variable name and type).

In order to generate the tests, the Test Generation module gets the extracted variables from the list of structs. Based on the chosen algorithm (base choice or random) given by the GUI, appropriate values are assigned to each variable depending on the given interval also specified by the GUI.

The Save to CSV takes the filename, data from the Test Generation presented in the GUI and saves the test in a file of CSV format.

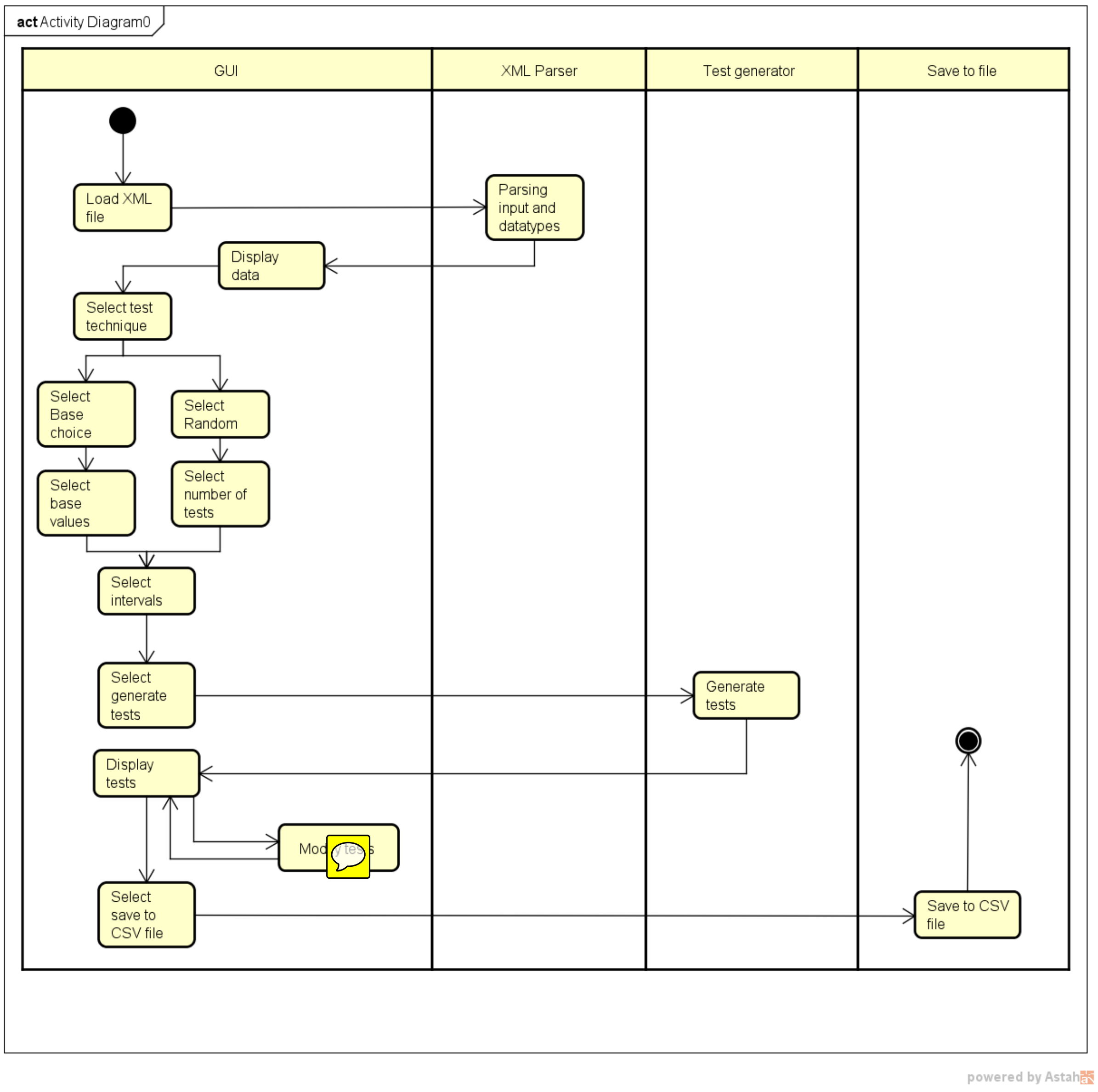
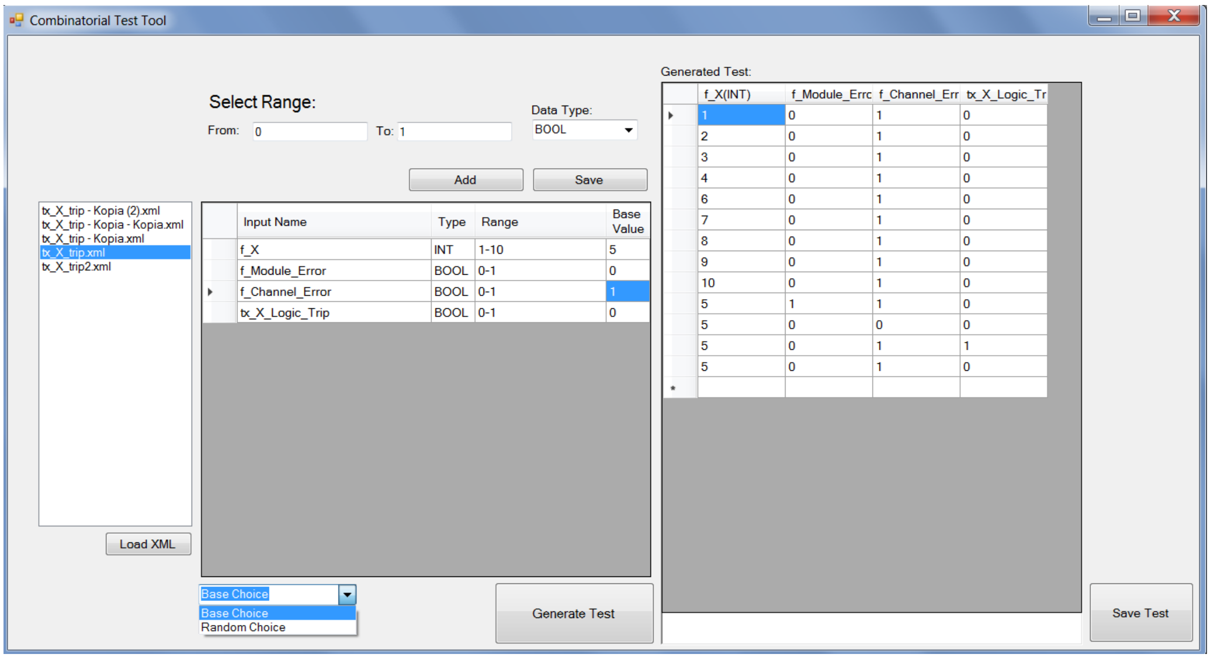


Figure 4. Activity diagram

6. GUI design



Input Name	Type	Range	Base Value
f_X	INT	1-10	5
f_Module_Error	BOOL	0-1	0
f_Channel_Error	BOOL	0-1	1
tx_X_Logic_Trip	BOOL	0-1	0

Generated Test:

f_X(INT)	f_Module_Errc	f_Channel_Err	tx_X_Logic_Tr
1	0	1	0
2	0	1	0
3	0	1	0
4	0	1	0
6	0	1	0
7	0	1	0
8	0	1	0
9	0	1	0
10	0	1	0
5	1	1	0
5	0	0	0
5	0	1	1
5	0	1	0

Figure 5. GUI design

To generate a test the user will first have to load which folder he wants to open. The content of this folder will be displayed in the small box to the far left of the tool. It is now possible to switch between which xml file from the selected folder that the user would want to generate tests from. As soon as an xml file is selected its contents is displayed in the gridview on the left side of the tool. Base choice is always the default test technique. If the user selects random from the dropdown menu instead of base choice, the base value column in the gridview will disappear, and a selection of what number of tests to be run will appear under the algorithm drop-down box. It is now possible to choose the interval range for each input, using either the select range fields to set an interval for all inputs of that type, or to directly type the interval or values that you want on a specific input into the gridview. Once the user is satisfied with all values and intervals for each input, the “Generate Test” button will generate test cases based on those intervals and inputs. This newly generated test is displayed in the view on the right side of the screen. Pressing the save button will create a csv file with the generated test and the same name as the original xml file.

There is an optional requirement of adding the possibility to move, remove or add test cases manually before saving to the csv. These button and functionality will more likely be added to the right side of the GUI. This would make the most sense due to the locality of where the test cases are displayed.