CS6474: Software Testing Laboratory
(Spring 2022)

Bishwajit Prasad Gond
222CS3113

Master of Technology
222cs3113@nitrkl.ac.in

Department of Computer Science & Engineering
NIT, Rourkela

April 11, 2023

CS6474: Software Testing Laboratory 2022

TCASES TOOL

Prepared by
BISHWAJIT PRASAD GOND
222CS3113

Contents

1 Tcases **1**

1.0.1 Tcases CMD command 1

1.1 Annotations-Test 1

1.1.1 Specification 1

1.1.2 XML Code 1

1.2 Ice-Cream Input 3

1.2.1 Specification 3

1.2.2 XML Code 3

1.3 Tcases-Input 6

1.3.1 Specification 6

1.3.2 Tcases XML Code 8

1.4 Find-Input 13

1.4.1 Specification 13

1.4.2 Find-Input XML Code 14

1 Tcases

Tcases is a tool for designing tests. It doesn't matter what kind of system you are testing. Nor does it matter what level of the system you are testing — unit, subsystem, or full system. You can use Tcases to design your tests in any of these situations. With Tcases, you define the input space for your system-under-test and the level of coverage that you want. Then Tcases generates a minimal set of test cases that meets your requirements.

Tcases is primarily a tool for black-box test design. For such tests, the concept of "coverage" is different from structural testing criteria such as line coverage, branch coverage, etc. Instead, Tcases is guided by coverage of the input space of your system.

1.0.1 Tcases CMD command

```
1 D:\Tcase\tcases-3.7.1\docs\examples\xml\icecream3>tcases Ice-
    Cream-Input.xml
2 D:\Tcase\tcases-3.7.1\docs\examples\xml\icecream3>tcases -J Ice-
    Cream-Input.xml
3 D:\Tcase\tcases-3.7.1\docs\examples\xml\icecream3>tcases -H Ice-
    Cream-Input.xml
4 D:\Tcase\tcases-3.7.1\docs\examples\xml\icecream3>tcases-reducer
    Ice-Cream-Input.xml
5 D:\Tcase\tcases-3.7.1\docs\examples\xml\icecream3>
```

1.1 Annotations-Test

1.1.1 Specification

Usage: Check different geometrical shape

We can add shape by using addShape function by passing parameter to
1) Type: CIRCLE, RECTANGLE, LINE etc
2) Size: In cm
3) Colour: RED, GREEN etc

1.1.2 XML Code

```
1 <!--
2 Usage: ADD or Create different geometrical shape
3
4 We can add shape by using addShape function
5 by passing parameter to
6 1) Type: CIRCLE, RECTANGLE, LINE etc
7 2) Size: In cm
8 3) Colour: RED, GREEN etc
```

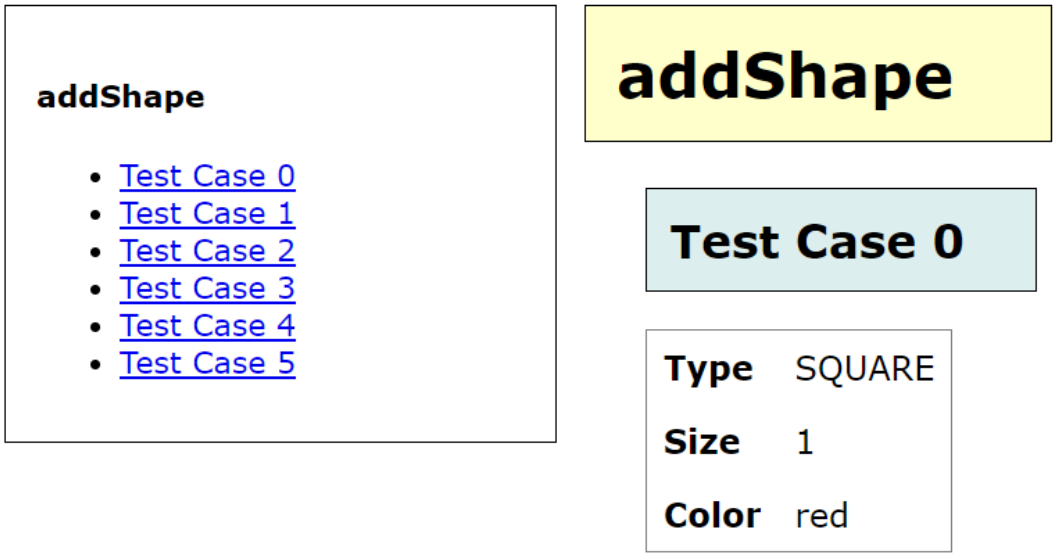

Figure 1: Annotations-Test HTML View

```
9
10  -->
11  <System name="Examples">
12      <Function name="addShape">
13          <!-- Test case annotations -->
14          <Has name="pageType" value="Page"/>
15          <Has name="pageName" value="page"/>
16          <Has name="pageValue" value="new Page()"/>
17
18          <Input>
19
20              <Var name="Type">
21                  <!-- Variable binding annotations -->
22                  <Has name="varType" value="Shape"/>
23                  <Has name="varName" value="shape"/>
24                  <Has name="varEval" value="new Shape"/>
25
26                  <Value name="SQUARE"/>
27                  <Value name="CIRCLE"/>
28                  <Value name="LINE" property="1D"/>
29              </Var>
30
31              <Var name="Size">
32                  <!-- Variable binding annotations -->
33                  <Has name="varType" value="int"/>
34                  <Has name="varName" value="size"/>
35                  <Has name="varApply" value="setSize"/>
36
37                  <Value name="1"/>
```

```
                <Value name="10"/>
                <Value name="100" property="Large"/>
            </Var>

            <Var name="Color">
                <!-- Variable binding annotations -->
                <Has name="varType" value="String"/>
                <Has name="varName" value="color"/>
                <Has name="varApply" value="setColor"/>

                <Value name="red"/>
                <Value name="green"/>
                <Value name="blue"/>
            </Var>
        </Input>
    </Function>
</System>
```

1.2 Ice-Cream Input

1.2.1 Specification

Usage: Checking different Scoop level & toppings of ice-cream

We can add different levels of scoops in different condition of cone functions
for example
1) If cone is not there then we will not add any scoop of ice-cream.
2) If cone is there and customer want Plain ice-cream then we can add maximum of 1 scoop and 1 topping in ice-cream.
3) If cone is there and customer want Plenty ice-cream then we can add min of 1 and maximum of 2 scoop and 2 topping in ice-cream.
4) If cone is there and customer want Grande ice-cream then we can add maximum of 4 scoop and 3 maximum and 1 minimum topping in ice-cream.
5) If cone is there and customer want Too-much ice-cream then we can add maximum of 3 scoop and 4 minimum topping in ice-cream.
6) If cone is there and customer want Too-much ice-cream then we can add maximum of 4 scoop and 5 minimum topping in ice-cream.

1.2.2 XML Code

```
<System name="Ice-Cream">
    <Function name="Cones">
        <Input>
            <Var name="Cone">
```

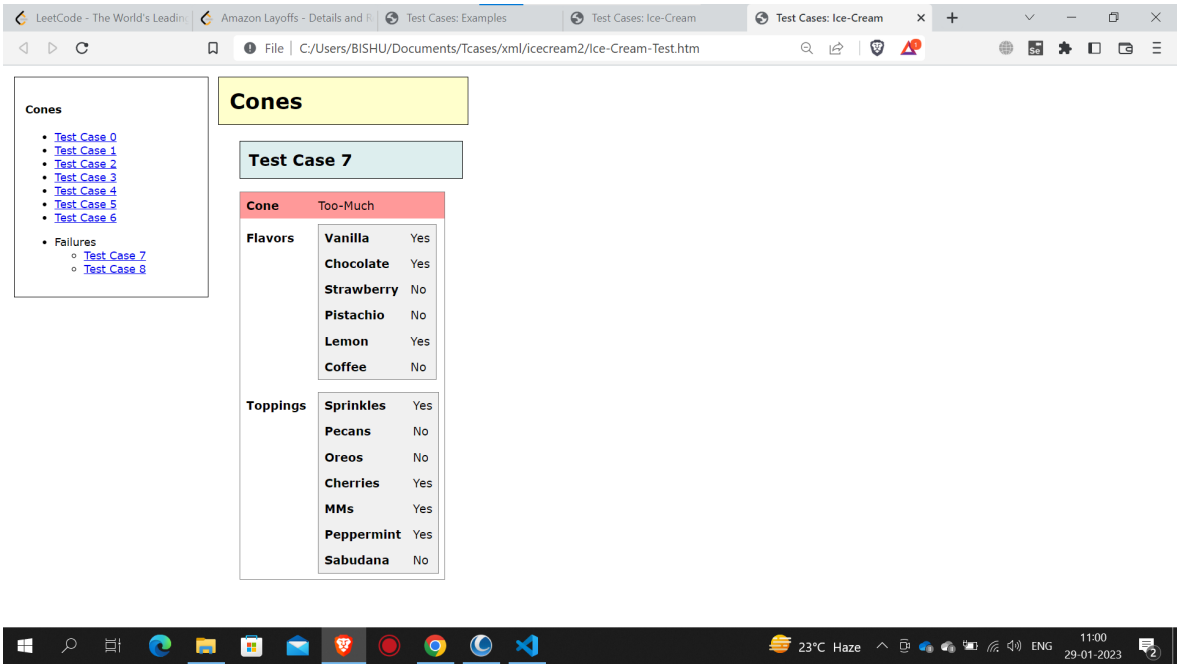

Figure 2: Ice-Cream HTML Test cases View

```
                <Value name="Empty" failure="false">
                    <When>
                        <LessThan property="scoop" max="1"/>
                    </When>
                </Value>
                <Value name="Plain">
                    <When>
                        <AllOf>
                            <Equals property="scoop" count="1"/>
                            <NotMoreThan property="topping" max="
                               1"/>
                        </AllOf>
                    </When>
                </Value>
                <Value name="Plenty">
                    <When>
                        <AllOf>
                            <Between property="scoop" min="1" max
                               ="2"/>
                            <NotMoreThan property="topping" max="
                               2"/>
                        </AllOf>
                    </When>
                </Value>
                <Value name="Grande">
                    <When>
                        <AllOf>
```

```
                        <Between property="scoop"
                            exclusiveMin="0" exclusiveMax="4"/
                            >
                        <Between property="topping" min="1"
                            max="3"/>
                    </AllOf>
                </When>
            </Value>
            <Value name="Too-Much" failure="true">
                <When>
                    <AnyOf>
                        <MoreThan property="scoop" min="3"/>
                        <NotLessThan property="topping" min="
                            4"/>
                    </AnyOf>
                </When>
            </Value>
            <Value name="Too-too-Much" failure="true">
                <When>
                    <AnyOf>
                        <MoreThan property="scoop" min="4"/>
                        <NotLessThan property="topping" min="
                            5"/>
                    </AnyOf>
                </When>
            </Value>
        </Var>

        <VarSet name="Flavors">
            <Var name="Vanilla">
                <Value name="Yes" property="scoop"/>
                <Value name="No"/>
            </Var>
            <Var name="Chocolate">
                <Value name="Yes" property="scoop"/>
                <Value name="No"/>
            </Var>
            <Var name="Strawberry">
                <Value name="Yes" property="scoop"/>
                <Value name="No"/>
            </Var>
            <Var name="Pistachio">
                <Value name="Yes" property="scoop"/>
                <Value name="No"/>
            </Var>
            <Var name="Lemon">
                <Value name="Yes" property="scoop"/>
                <Value name="No"/>
```

```
                </Var>
                <Var name="Coffee">
                    <Value name="Yes" property="scoop"/>
                    <Value name="No"/>
                </Var>
            </VarSet>

            <VarSet name="Toppings" when="scoop">
                <Var name="Sprinkles">
                    <Value name="Yes" property="topping"/>
                    <Value name="No"/>
                </Var>
                <Var name="Pecans">
                    <Value name="Yes" property="topping"/>
                    <Value name="No"/>
                </Var>
                <Var name="Oreos">
                    <Value name="Yes" property="topping"/>
                    <Value name="No"/>
                </Var>
                <Var name="Cherries">
                    <Value name="Yes" property="topping"/>
                    <Value name="No"/>
                </Var>
                <Var name="MMs">
                    <Value name="Yes" property="topping"/>
                    <Value name="No"/>
                </Var>
                <Var name="Peppermint">
                    <Value name="Yes" property="topping"/>
                    <Value name="No"/>
                </Var>
                <Var name="Sabudana">
                    <Value name="Yes" property="topping"/>
                    <Value name="No"/>
                </Var>
            </VarSet>
        </Input>
    </Function>
</System>
```

1.3 Tcases-Input

1.3.1 Specification

Usage: Represents a set of command line options. Command line arguments have the following form. [-c tupleSize] [-f outFile] [-g genDef] [-n] [-I] [-o outDir] [-p name=value] [-r seed] [-R]

[-t testDef] [-T contentType] [-v] [-x transformDef — -J — -H] [inputDef]

where:
-c tupleSize If -c is defined, use the given default tupleSize for all generators. This updates the generator definitions specified by the genDef file.

-f outFile If -f is defined, test definition output is written to the specified outFile, relative to the given outDir. If omitted, test definitions are written to the file specified by the -t option. If an output path cannot be derived, output is written to standard output.

-g genDef If -g is defined, test definitions are created using the generator(s) specified by the given genDef file. If omitted, the default generator definition is used. The default generator definition is read from the corresponding *-Generators file in the same directory as the inputDef, if it exists. Otherwise, the default TupleGenerator is used for all functions.

-H If -H is defined, test definition output is transformed into an HTML report. The resulting HTML file is written to the specified outDir.

-J If -J is defined, test definition output is transformed into Java source code for a JUnit test class. The resulting Java source file is written to the specified outDir. The following parameters (see the -p option) affect the results of this transform.

class: The name of the class under test. If omitted, the default is defined by the system parameter.

system: The name of the system under test. If omitted, the default is defined by the inputDef.

throws: A boolean value (true/false, yes/no). If true, generated test methods are declared to throw Exception. If omitted, the default is false.

values: A boolean value (true/false, yes/no). If true, comments listing all input variable assignments are included in the body of each test method. If omitted, the default is true.

-n If -n is defined, any previous contents of the testDef are ignored. If omitted, new test definitions are based on the previous testDef.

-I If -I is defined, no test definitions are produced. Instead, a JSON document containing the effective system input definition is written to the outDir. The effective system input definition, which is used to generate test definitions, is the result of normalizing all input schemas and adding any schema-derived value definitions.

-o outDir If -o is defined, test definition output is written to the specified directory. If omitted, the default outDir is the directory containing the inputDef or, if reading from standard input, the current working directory. If an output path cannot be derived, output is written to standard output.

-p name=value Defines the value of a transform parameter. Any number of -p options may

be specified. This option is meaningful only if the -x or -J option is given.

-r seed If -r is defined, use the given random number seed for all generators. This updates the generator definitions specified by the genDef file. -R Choose a new random number seed for all generators. This updates the generator definitions specified by the genDef file.

-x transformDef If -x is defined, test definition output is transformed according to the XSLT transform defined by the transformDef file. If relative, the transformDef path is assumed to be relative to the directory containing the inputDef.

-t testDef If -t is defined, new test definitions are based on the contents of the specified testDef file, relative to the directory containing the inputDef. Also, unless the -f option is given, new test definition output is written to the specified testDef path, relative to the outDir. If omitted, the default testDef name is derived from the inputDef name.

-T contentType Defines the default content type for the files read and produced. The contentType must be one of ”json” or ”xml”. The default content type is assumed for any file that is not specified explicitly or that does not have a recognized extension. If omitted, the default content type is derived from the inputDef name.

-v Prints the current Tcases version identifier to standard output.

inputDef The system input definition is read from the given inputDef. If omitted, the system input definition is read from standard input. Otherwise, the system input definition is read from the first one of the following files that can be located.
inputDef
inputDef-Input.xml
inputDef.xml
inputDef-Input.json
inputDef.json

1.3.2 Tcases XML Code

```
<System name="Tcases">
  <Function name="run">
    <Input>

      <!-- Option: -c -->
      <VarSet name="defaultTupleSize">
        <Var name="defined">
          <Value name="Yes" property="defaultTupleSize"/>
          <Value name="No"/>
        </Var>

        <Var name="isNumber" when="defaultTupleSize">
```

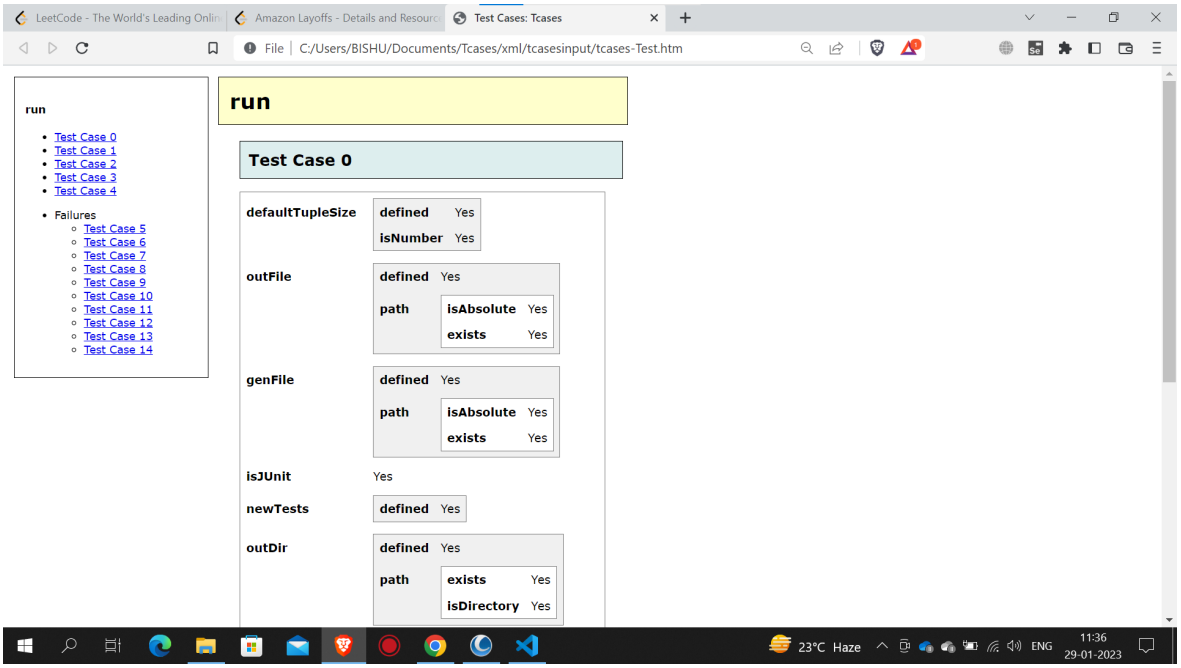

Figure 3: TCases Html View

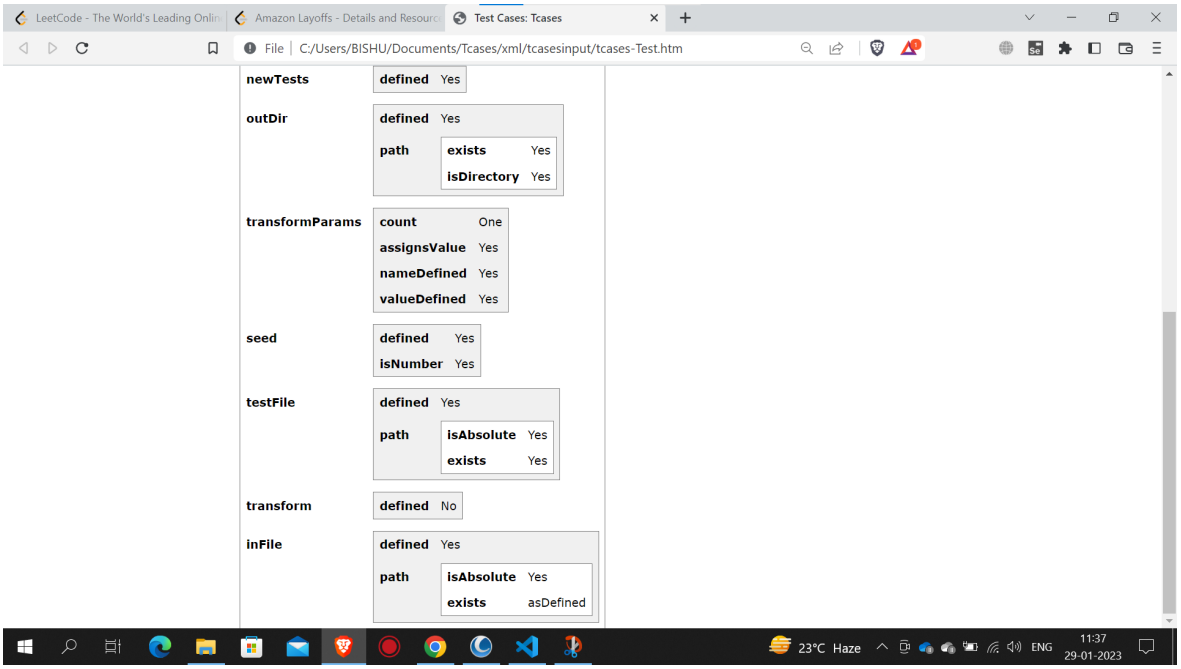

Figure 4: TCases Html View

```
        <Value name="Yes"/>
        <Value name="No" failure="true"/>
      </Var>
    </VarSet>

    <!-- Option: -f -->
    <VarSet name="outFile">
      <Var name="defined">
```

```
21          <Value name="Yes" property="outFile"/>
22          <Value name="No"/>
23          <Value name="TransformOutputUndefined" failure="true"
               when="transformedOut, testFileExists"/>
24        </Var>
25
26        <VarSet name="path" when="outFile">
27          <Var name="isAbsolute">
28            <Value name="Yes"/>
29            <Value name="No"/>
30          </Var>
31          <Var name="exists">
32            <Value name="Yes" property="outFileExists"/>
33            <Value name="No"/>
34          </Var>
35        </VarSet>
36      </VarSet>
37
38      <!-- Option: -g -->
39      <VarSet name="genFile">
40        <Var name="defined">
41          <Value name="Yes" property="genFile"/>
42          <Value name="No"/>
43        </Var>
44
45        <VarSet name="path" when="genFile">
46          <Var name="isAbsolute">
47            <Value name="Yes"/>
48            <Value name="No"/>
49          </Var>
50          <Var name="exists">
51            <Value name="Yes"/>
52            <Value name="No" failure="true"/>
53          </Var>
54        </VarSet>
55
56        <Var name="default" whenNot="genFile">
57          <Value name="ForInputExists" when="inFile"/>
58          <Value name="ForInputNone" when="inFile"/>
59          <Value name="Standard" whenNot="inFile"/>
60        </Var>
61      </VarSet>
62
63      <!-- Option: -J -->
64      <Var name="isJUnit">
65        <Value name="Yes" property="isJUnit, transformedOut"
             whenNot="transform"/>
66        <Value name="No" when="transform"/>
```

```
67          <Value name="NotAllowed" when="transform" failure="true"/
                >
68        </Var>
69
70        <!-- Option: -n -->
71        <VarSet name="newTests">
72          <Var name="defined" when="testFileExists">
73            <Value name="Yes"/>
74            <Value name="No"/>
75          </Var>
76        </VarSet>
77
78        <!-- Option: -o -->
79        <VarSet name="outDir">
80          <Var name="defined">
81            <Value name="Yes" property="outDir"/>
82            <Value name="No"/>
83          </Var>
84
85          <VarSet name="path" when="outDir">
86            <Var name="exists">
87              <Value name="Yes" property="outDirExists"/>
88              <Value name="No"/>
89            </Var>
90            <Var name="isDirectory" when="outDirExists">
91              <Value name="Yes"/>
92              <Value name="No" failure="true"/>
93            </Var>
94          </VarSet>
95        </VarSet>
96
97        <!-- Option: -p -->
98        <VarSet name="transformParams">
99          <When>
100           <AnyOf property="transform, isJUnit"/>
101         </When>
102         <Var name="count">
103           <Value name="One" property="params"/>
104           <Value name="Many" property="params"/>
105           <Value name="None"/>
106         </Var>
107         <Var name="assignsValue" when="params">
108           <Value name="Yes"/>
109           <Value name="No" failure="true"/>
110         </Var>
111         <Var name="nameDefined" when="params">
112           <Value name="Yes"/>
113           <Value name="No" failure="true"/>
```

```
      </Var>
      <Var name="valueDefined" when="params">
        <Value name="Yes"/>
        <Value name="No"/>
      </Var>
    </VarSet>

    <!-- Option: -r -->
    <VarSet name="seed">
      <Var name="defined">
        <Value name="Yes" property="random"/>
        <Value name="No"/>
      </Var>

      <Var name="isNumber" when="random">
        <Value name="Yes"/>
        <Value name="No" failure="true"/>
      </Var>
    </VarSet>

    <!-- Option: -t -->
    <VarSet name="testFile">
      <Var name="defined">
        <Value name="Yes" property="testFile"/>
        <Value name="No"/>
      </Var>

      <VarSet name="path" when="testFile">
        <Var name="isAbsolute">
          <Value name="Yes"/>
          <Value name="No"/>
        </Var>
        <Var name="exists">
          <Value name="Yes" property="testFileExists"/>
          <Value name="No"/>
        </Var>
      </VarSet>

      <VarSet name="default" whenNot="testFile">
        <Var name="exists">
          <Value name="Yes" property="testFileExists"/>
          <Value name="No"/>
        </Var>
      </VarSet>
    </VarSet>

    <!-- Option: -x -->
    <VarSet name="transform">
```

```
        <Var name="defined">
          <Value name="Yes" property="transform, transformedOut"/
             >
          <Value name="No"/>
        </Var>
        <VarSet name="path" when="transform">
          <Var name="isAbsolute">
            <Value name="Yes"/>
            <Value name="No"/>
          </Var>
          <Var name="exists">
            <Value name="Yes"/>
            <Value name="No" failure="true"/>
          </Var>
        </VarSet>
      </VarSet>

      <!-- Input definition file -->
      <VarSet name="inFile">
        <Var name="defined">
          <Value name="Yes" property="inFile"/>
          <Value name="No"/>
        </Var>
        <VarSet name="path" when="inFile">
          <Var name="isAbsolute">
            <Value name="Yes"/>
            <Value name="No"/>
          </Var>
          <Var name="exists">
            <Value name="asDefined"/>
            <Value name="withInputXml"/>
            <Value name="withXml"/>
            <Value name="No" failure="true"/>
          </Var>
        </VarSet>
      </VarSet>

    </Input>
  </Function>
</System>
```

1.4 Find-Input

1.4.1 Specification

Usage: find pattern file

Locates one or more instances of a given pattern in a text file.

All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file. To include a blank in the pattern, the entire pattern must be enclosed in quotes (”). To include a quotation mark in the pattern, two quotes in a row (””) must be used.

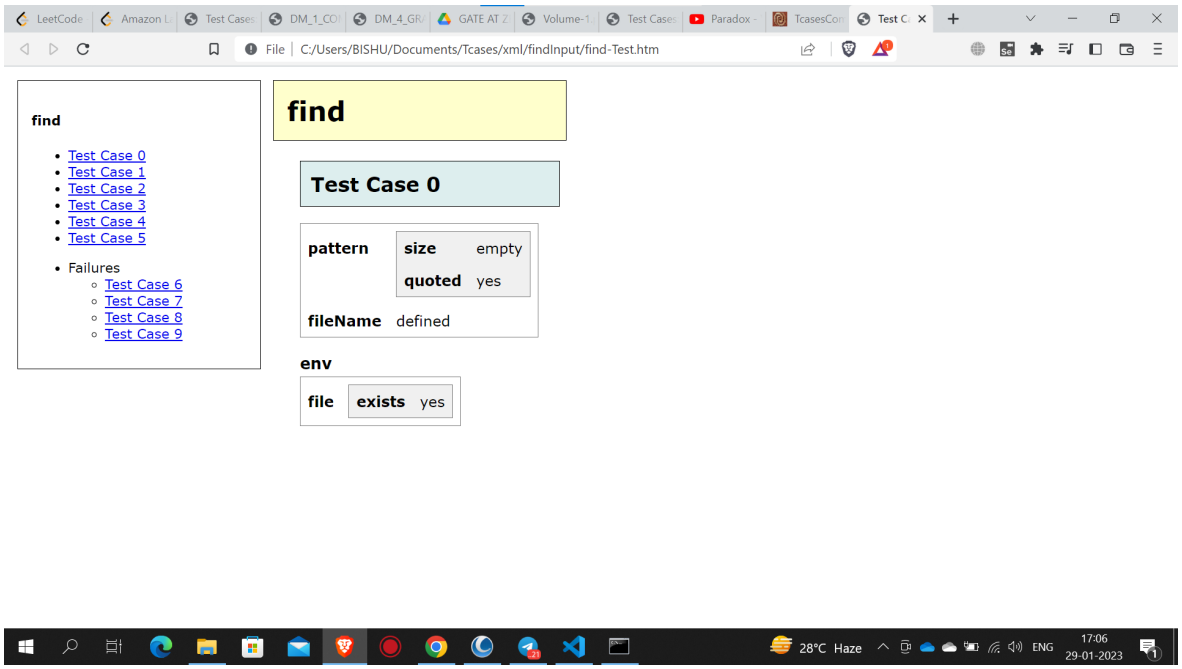

Figure 5: Find-Input HTML View

1.4.2 Find-Input XML Code

```
1
2  <System name="Examples">
3    <Function name="find">
4      <!--
5           Usage: find pattern file
6
7           Locates one or more instances of a given pattern in a
               text file.
8
9           All lines in the file that contain the pattern are
               written to standard output. A
10          line containing the pattern is written only once,
               regardless of the number of
11          times the pattern occurs in it.
12
13          The pattern is any sequence of characters whose length
               does not exceed the
```

```
14          maximum length of a line in the file. To include a blank
                in the pattern, the
15          entire pattern must be enclosed in quotes ("). To
               include a quotation mark in the
16          pattern, two quotes in a row ("") must be used.
17        -->
18      <Input type="arg">
19        <VarSet name="pattern" when="fileExists">
20          <Var name="size">
21            <Value name="empty" property="empty"/>
22            <Value name="singleChar" property="singleChar"/>
23            <Value name="manyChars"/>
24          </Var>
25          <Var name="quoted">
26            <Value name="yes" property="quoted"/>
27            <Value name="no" whenNot="empty"/>
28            <Value name="unterminated" failure="true"/>
29          </Var>
30          <Var name="blanks" whenNot="empty">
31            <Value name="none"/>
32            <Value name="one" when="quoted, singleChar"/>
33            <Value name="many">
34              <When>
35                <AllOf property="quoted">
36                  <Not property="singleChar"/>
37                </AllOf>
38              </When>
39            </Value>
40          </Var>
41          <Var name="embeddedQuotes" whenNot="empty, singleChar">
42            <Value name="none"/>
43            <Value name="one"/>
44            <Value name="many" once="true"/>
45          </Var>
46        </VarSet>
47
48        <Var name="fileName">
49          <Value name="defined" property="fileName"/>
50          <Value name="missing" failure="true"/>
51        </Var>
52      </Input>
53
54      <Input type="env">
55        <VarSet name="file" when="fileName">
56          <Var name="exists">
57            <Value name="yes" property="fileExists"/>
58            <Value name="no" failure="true"/>
59          </Var>
```

```
60          <VarSet name="contents" when="fileExists" whenNot="empty"
               >
61            <Var name="linesLongerThanPattern">
62              <Value name="one" property="matchable" once="true"/>
63              <Value name="many" property="matchable"/>
64              <Value name="none" failure="true"/>
65            </Var>
66            <Var name="patterns" when="matchable" whenNot="empty">
67              <Value name="none" once="true"/>
68              <Value name="one" property="match"/>
69              <Value name="many" property="match, many"/>
70            </Var>
71            <Var name="patternsInLine" when="match">
72              <Value name="one"/>
73              <Value name="many" once="true" when="many"/>
74            </Var>
75          </VarSet>
76        </VarSet>
77      </Input>
78
79    </Function>
80  </System>
```