Experiment No 1

Aim: Find out the Roots of a Quadratic Equation and Generate its Boundary Value Test Cases.

Input: A quadratic equation $a(x^2)+bx+c=0$ with input as three positive integers a, b, c having values ranging from an interval [0,100].

Theory:

Consider a program for the determination of the nature of roots of a quadratic equation. Its input is a triple of positive integers (Say a, b, c) and values may be from interval [0, 100]. The program output may have one of the following words. [Not a quadratic equation Real roots; Imaginary roots, Equal roots] Quadratic Equation will be of type:

ax2 + bx + c = 0

Roats are real if (b2 – 4 ac) > 0

Roots are imaginary if (b2 – 4 ac) < 0

Roots are equal if (b2 – 4 ac) = 0

Equation is not quadratic if a = 0

Program

```
#include<iostream.h>

#include<conio.h>

void main( )

{

clrscr( );

int a,b,c,d;

cout<<"The quadratic equation is of the type a(x2)+bx+c=0"

<cout<<"Enter the value of a:"<cin>>a;
cout<<"Enter the value of b:"<cin>>b;

cout<<"Enter the value of c: "<cin>>c;

d = (b*b)-4*a*c;

if((a<0)||(b<0)||(c<0)||(a>100)||(b>100)||(c>100))

cout<<"Invalid input"<<end1;

elseif(a==0)

cout<<"Not a quadratic equation"<elseif(d==0)

cout<<"Roots are equal"<else if(d<0)

cout<<"Imaginary roots"<else

cout<<"Real roots"<getch();

}
```

Boundary Value Analysis:
Total number of Test cases = $4n + 1$
where $n \rightarrow$ number of inputs= 4(3) + 1 = 13 Test cases
Boundary value test cases are given as:
In the above program, we consider the values as 0 (Minimum), 1 (Just above Minimum), 50 (Nominal), 99 (Just below Maximum) and 100 (Maximum).

Test Case ID	a	b	c	Expected Output
1	50	50	0	Real Roots
2	50	50	1	Real Roots
3	50	50	50	Imaginary Roots
4	50	50	99	Imaginary Roots

5	50	50	100	Imaginary Roots
6	50	0	50	Imaginary Roots
7	50	1	50	Imaginary Roots
8	50	99	50	Imaginary Roots
9	50	100	50	Equal Roots
10	0	50	50	Not a Quadratic Equation
11	1	50	50	Real Roots
12	99	50	50	Imaginary Roots
13	100	50	50	Imaginary Roots

Boundary Value Analysis focuses on the input variables of the function. For the purposes of this report I will define two variables (I will only define two so that further examples can be kept concise) X1 and X2. Where X1 lies between A and B and X2 lies between C and D. $A \le X1 \le B$ $C \le X2 \le D$.

- In the general application of Boundary Value Analysis can be done in a uniform manner. The basic form of implementation is to maintain all but one of the variables at their nominal (normal or average) values and allowing the remaining variable to take on its extreme values. The values used to test the extremities are:
  • Min ----------------------------------- - Minimal

  • Min+ ----------------------------------- - Just above Minimal

  • Nom ----------------------------------- - Average

• Max- ----------------------------------- - Just below Maximum

• Max ------------------------------------ - Maximum

Some Important examples:

The NextDate problem

1 ≤ Day ≤ 31.

1 ≤ month ≤ 12.

1812 ≤ Year ≤ 2012.

(Here the year has been restricted so that test cases are not too large).

Conclusion: Thus we have studied and executed program to **Find out the Roots of a Quadratic Equation and executed its Boundary Value Test Cases.**

Experiment No 2

Aim: Write a program in C/C++ to find the area of a circle, triangle, square and rectangle and perform Equivalence Class testing

Theory: In Equivalence Class Testing, we find two types of equivalence classes

1) Input Domain and

2) Output Domain.

Input Domain is formed from one valid sequence and two invalid sequences. The Output Domain is obtained from different types of outputs of the problem.

Program in C++

```
 size=2>

#include

#include

#include

void main()

{

clrscr();

int ch;

char c;

float, b, h, a;

1: cout<<"Enter your choice";

cout<<"n1.Triangle";

cout<<"n2.Square";

cout<<"n3.Rectangle";


cout<<"n4.Circle";

cout<<"n5.Exitn";

cin>>ch;

switch(ch)

{ case 1 : b: cout<<"nEnter the base of the triangle (1-200)";

cin>>b;
```

```
if ((b<=0)||(b>200))

{ cout<<"nInvalid entry for base n";

goto b;

}

h: cout<<"nEnter the height of the triangle (1-200)";

cin>>h;

if ((h<=0)||(h>200))

{ cout<<"nInvalid height nEnter the height (1-200)";

goto h;

}

a= 0.5*b*h;

cout<<"nThe area is "<

cout<<"nWant to enter more?(y/n) ";

cin>>c;

if((c=='y')||(c=='Y'))

goto 1;

break

case 2 : s: cout<<"nEnter the side of the square (1-200)";


cin>>b;

if ((b<=0)||(b>200))

{ cout<<"nInvalid entry for base n";

goto s;

}

a= b*b;
```

```
cout<<"nThe area is "<

cout<<"nWant to enter more?(y/n) ";

cin>>c;

if((c=='y')||(c=='Y'))

goto 1;

break;

case 3: d: cout<<"nEnter the base of the triangle (1-200)" ;

cin>>b;

if((b<=0)||(b>200))

( cout<<"nInvalid entry for base n";

goto d;

}

p: cout<<"nEnter the height of the triangle (1-200) ";

cin>>h;

if ((h<=0)||(h>200))

{ cout<<"nInvalid height nEnter the height(1-200)";

goto p;

}


a=b*h;

cout<<"nThe area is "<

cout<<"nWant to enter more?(y/n) ";

cin>>c;

if((c=='y')||(c=='Y'))

goto 1;
```

```
break;
case 4: t: cout<<"nEnter the radius of the circle ";
cin>>b;
if ((b<=0)||(b>200))
{ cout<<"nInvalid entry for base n";
goto t;
}
a= 3.14*b*b;
cout<<"nThe area is "<
cout<<"nWant to enter more?(y/n)";
cin>>c;
if ((c=='y')||(c=='Y'))
goto 1;
break;
case 5: exit(0);
break;
default : cout<<"n WRONG CHOICE";
goto 1;


 }
getch();
}
```

1. Triangle :

I1 = {h : h<=0}

I2 = {h : h>200}

Test Case ID	h	b	Expected Output
1	0	100	Invalid Input
2	100	100	5000
3	201	100	Invalid Input
4	100	0	Invalid Input
5	100	100	5000

I3 = {h : 1<=h<=200}

I4 = {b : b<=0}

I5 = {b : b>200}

I6 = {b : 1<=b<=2001}

Output Domain:

O1 = {: Triangle if h > 0, b > 0}

O2 = {: Not a triangle if h <= 0, b <= 0}

2. Square

Input Domain:

I1 = {s : s<=0}

I2 = {s : s>200}

I3 = {s : 1<=s<=200}

Test Case ID	s	Expected Output
1	0	Invalid Input
2	100	10000
3	201	Invalid Input

Output Domain:

O1 = {: Square if s>0}

O2 = {: Not a square if s <= 0}

Rectangle

Input Domain:

I1 = {l : l<=0}

I2 = {l : l>200}

I3 = {l : 1<=l<=200}

Test Case ID	l	b	Expected Output
1	0	100	Invalid Input
2	100	100	10000
3	201	100	Invalid Input
4	100	0	Invalid Input
5	100	100	10000
6	100	201	Invalid Input

I4 = {b : b<=0}

I5 = {b : b>200}

I6 = {b : 1<=b<=200}

Circle

Input Domain:

I1 = {r : r<=0}

I2 = {r : r>200}

I3 = {r : 1<=r<=200}

Test Case ID	r	Expected Output
1	0	Invalid Input
2	100	31400
3	201	Invalid Input

Output Domain:

O1 = {: Circle if 1<=r<=200}

O2 = {: Not a Circle if r <= 0}

Conclusion: Thus we have studied and executed program to find the area of a circle, triangle, square and rectangle **and performed Equivalence Class testing with sample test cases.**

Experiment No 3

Aim: Write Test Cases for any one given Application.

Theory: A Test Case is a set of actions executed to verify a particular feature or functionality of your software application.

A **TEST CASE** is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application.

Test Case Template

A test case can have the following elements. Note, however, that a test management tool is normally used by companies and the format is determined by the tool used.

Test Suite ID	The ID of the test suite to which this test case belongs.
Test Case ID	The ID of the test case.
Test Case Summary	The summary / objective of the test case.
Related Requirement	The ID of the requirement this test case relates/traces to.
Prerequisites	Any prerequisites or preconditions that must be fulfilled prior to executing the test.
Test Procedure	Step-by-step procedure to execute the test.
Test Data	The test data, or links to the test data, that are to be used while conducting the test.
Expected Result	The expected result of the test.
Actual Result	The actual result of the test; to be filled after executing the test.
Status	Pass or Fail. Other statuses can be ‘Not Executed’ if testing is not performed and ‘Blocked’ if testing is blocked.
Remarks	Any comments on the test case or test execution.
Created By	The name of the author of the test case.

Date of Creation	The date of creation of the test case.
Executed By	The name of the person who executed the test.
Date of Execution	The date of execution of the test.
Test Environment	The environment (Hardware/Software/Network) in which the test was executed.

Writing Good Test Cases

- As far as possible, write test cases in such a way that you test only one thing at a time. Do not overlap or complicate test cases. Attempt to make your test cases 'atomic'.
- Ensure that all positive scenarios AND negative scenarios are covered.
- Language:
  - Write in simple and easy-to-understand language.
  - Use active voice instead of passive voice: Do this, do that.
  - Use exact and consistent names (of forms, fields, etc).
- Characteristics of a good test case:
  - *Accurate*: Exacts the purpose.
  - *Economical*: No unnecessary steps or words.
  - *Traceable*: Capable of being traced to requirements.
  - *Repeatable*: Can be used to perform the test over and over.
  - *Reusable*: Can be reused if necessary.

Conclusion: We have Configured a simple web application on local host and designed Test Cases for given Application and Tested Application Successfully.

Experiment No 4

Aim: Write a program in C/C++ to read 3 sides of a triangle & to determine whether they form scalene, isosceles or equilateral triangle and test the same using basis path testing and find its V(G) by all the three methods.

Theory: Basis path testing helps a tester to compute logical complexity measure, V (G), of the code. This value of V (G), defines the maximum number of test cases to be designed by identifying basis set of execution paths to ensure that all statements are executed at least once.

Steps to compute the complexity measure, V (G) are as under

Step–1: Construct the flow graph from the source code or flow charts.
Step –2: Identify independent paths.
Step–3: Calculate Cyclomatic Complexity, V (G).
Step –4: Design the test cases.

Draw Flow chart -Convert into Directed Graph-Here

Calculation of Cyclomatic Complexity V(G) by three methods:

Method – 1: V(G) = e – n + 2 (Where "e" are edges & "n" are nodes)

V(G)=10–8+2=2+2=4

Method – 2: V(G) = P + 1 (Where P – No. of predicate nodes with out degree = 2)

V(G) = 3+ 1 = 4 (Nodes 2, 4 & 6 are predicate nodes with 2 outgoing edges)

Method–3: V(G)=Number of enclosed regions+1=3+1=4
(Here R1, R2 & R3 are the enclosed regions and 1 corresponds to one outer region)

V(G) = 4 and is same by all the three methods.

The test cases for each of the path are:

Test Case	valid input	Expected results
1 Enlist 1st Path	a, b, c : valid input	if s = b or b = c or a = c then message `isosceles

		triangle' is displayed.
2 Enlist 2nd Path	a, b, c : valid input	Expected results : if a≠b≠c then message `scalene triangle, is displayed.
3 Enlist 3rd Path	a, b, c : valid input	Expected results : if a=b=c then message `Equilateral triangle, is displayed.
4 Enlist 4th Path	a, b, c : invalid input	Go to Enter Values of a,b,c.

Conclusion: Thus we have studied and executed program to read 3 sides of a triangle & to determine whether they form scalene, isosceles or equilateral triangle and tested the same using basis path testing calculated V(G) by all the three methods.

Experiment No 5

Aim: Test a website using automation tool like QTP.

Theory:

Experiment No 6

Aim: Use any automated test tool.(e.g. Autoit V3 tool) and demonstrate the use of 1) If….else 2) For …Loop 3) Do…Until 4) Switch … Case.

Theory: AutoIt v3 is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. It uses a combination of simulated keystrokes, mouse movement and window/control manipulation in order to automate tasks in a way not possible or reliable with other languages.

AutoIt was initially designed for PC "roll out" situations to reliably automate and configure thousands of PCs. Over time it has become a powerful language that supports complex expressions, user functions, loops and everything else that veteran scripters would expect. Features:

- Easy to learn BASIC-like syntax
- Simulate keystrokes and mouse movements
- Manipulate windows and processes
- Interact with all standard windows controls
- Scripts can be compiled into standalone executables
- Create Graphical User Interfaces (GUIs)
- COM support
- Regular expressions
- Directly call external DLL and Windows API functions
- Scriptable RunAs functions
- Detailed helpfile and large community-based support forums
- Compatible with Windows XP SP3 / 2003 SP2 / Vista / 2008 / Windows 7 / 2008 R2 / Windows 8 / 2012 R2
- Unicode and x64 support
- Digitally signed for peace of mind
- Works with Windows User Account Control (UAC)

Use:

1) If….else

If <expression> Then *statement*

```
#include <MsgBoxConstants.au3>

Local $iNumber = -20
```

```
If $iNumber > 0 Then
                MsgBox($MB_SYSTEMMODAL, "Example", "$iNumber          was          positive!")
ElseIf $iNumber < 0 Then
                MsgBox($MB_SYSTEMMODAL, "Example", "$iNumber          was          negative!")
Else
                MsgBox($MB_SYSTEMMODAL, "Example", "$iNumber          was          zero.")
EndIf
```

2) For …Loop

For <variable>=<start> To <stop>[Step <stepval>]
***statements*……..**
Next

A For loop will execute zero times if:
** *start* > *stop* and *step* ≥ 0, or**
** *start* < *stop* and *step* is negative**

```
#include <MsgBoxConstants.au3>


For $i = 5 To 1 Step -1

   MsgBox($MB_SYSTEMMODAL, "", "Count down!" & @CRLF & $i)

Next

MsgBox($MB_SYSTEMMODAL, "", "Blast Off!")
```

3) Do…Until

Do
statements
...
Until <expression>

Do...Until statements may be nested.
The expression is tested after the loop is executed, so the loop will be executed one or more times.

```
#include <MsgBoxConstants.au3>


Local $i = 0

Do

    MsgBox($MB_SYSTEMMODAL, "", "The value of $i is: " & $i) ; Display the value of $i.

    $i = $i + 1 ; Or $i += 1 can be used as well.

Until $i = 10 ; Increase the value of $i until it equals the value of 10.
```

4) Switch … Case

Switch <expression>
Case <value> [To <value>] [,<value> [To <value>] ...]
statement1
...
[**Case** <value> [To <value>] [,<value> [To <value>] ...]
statement2
...]
[**Case** Else
statementN
...]
EndSwitch

If no cases match the Switch value, then the Case Else section, if present, is executed. If no cases match and Case Else is not defined, then none of the code inside the Switch structure, other than the initial expression, will be executed.

Switch statements may be nested. Switch statements are case-insensitive.

```
#include <MsgBoxConstants.au3>


Local $sMsg = ""
```

```
Switch @HOUR
    Case 6 To 11
        $sMsg = "Good Morning"
    Case 12 To 17
        $sMsg = "Good Afternoon"
    Case 18 To 21
        $sMsg = "Good Evening"
    Case Else
        $sMsg = "What are you still doing up?"
EndSwitch

MsgBox($MB_SYSTEMMODAL, "", $sMsg)
```

Conclusion: Thus we have installed and Configured Tool Autoit v3 and demonstrated use of **1) If….else 2) For …Loop 3) Do…Until 4) Switch … Case.**

Experiment No 7

Aim: Create any GUI Application e.g. Calculator and Automate using Autoit V3 tool.

Theory:

AutoIt is constantly evolving as a programming language. It started as an add-on tool to automate basic tasks in GUI's of other programs, and task automation (such as sending a keystroke or clicking a button) is still at the heart of AutoIt. With the introduction of many new features, however, AutoIt has become a more powerful tool than ever before.

Just a few of the new and updated features include:

- GUI Automation - Create a custom graphical interface for your application.
- COM (Object) functionality fills the gap with WSH languages such as VBScript/JScript.
- Loops, functions and expression parsing.
- An enormous number of functions for handling and manipulating Strings.
- A Perl-compatible regular expression engine using the PCRE library, with native 16bit mode and UCP/UTF support.
- A powerful Recursive File List to Array function
- Easily call Win32 and third-party DLL APIs from within your script.

Calc GUI:

```
#include <EditConstants.au3>

#include <GUIConstantsEx.au3>

#include <StaticConstants.au3>

#include <WindowsConstants.au3>

GUICreate("Calculator", 260, 230)

$idBtn1 = GUICtrlCreateButton("1", 54, 138, 36, 29)

; Digit's buttons

Local $idBtn0 = GUICtrlCreateButton("0", 54, 171, 36, 29)

Local $idBtn1 = GUICtrlCreateButton("1", 54, 138, 36, 29)

Local $idBtn2 = GUICtrlCreateButton("2", 93, 138, 36, 29)

Local $idBtn3 = GUICtrlCreateButton("3", 132, 138, 36, 29)

Local $idBtn4 = GUICtrlCreateButton("4", 54, 106, 36, 29)
Local $idBtn5 = GUICtrlCreateButton("5", 93, 106, 36, 29)

Local $idBtn6 = GUICtrlCreateButton("6", 132, 106, 36, 29)

Local $idBtn7 = GUICtrlCreateButton("7", 54, 73, 36, 29)

Local $idBtn8 = GUICtrlCreateButton("8", 93, 73, 36, 29)

Local $idBtn9 = GUICtrlCreateButton("9", 132, 73, 36, 29)

Local $idBtnPeriod = GUICtrlCreateButton(".", 132, 171, 36, 29)

; Memory's buttons

Local $idBtnMClear = GUICtrlCreateButton("MC", 8, 73, 36, 29)

Local $idBtnMRestore = GUICtrlCreateButton("MR", 8, 106, 36, 29)

Local $idBtnMStore = GUICtrlCreateButton("MS", 8, 138, 36, 29)

Local $idBtnMAdd = GUICtrlCreateButton("M+", 8, 171, 36, 29)

; Operators

Local $idBtnChangeSign = GUICtrlCreateButton("+/-", 93, 171, 36, 29)

Local $idBtnDivision = GUICtrlCreateButton("/", 171, 73, 36, 29)

Local $idBtnMultiplication = GUICtrlCreateButton("*", 171, 106, 36, 29)

Local $idBtnSubtract = GUICtrlCreateButton("-", 171, 138, 36, 29)

Local $idBtnAdd = GUICtrlCreateButton("+", 171, 171, 36, 29)

Local $idBtnAnswer = GUICtrlCreateButton("=", 210, 171, 36, 29)

Local $idBtnInverse = GUICtrlCreateButton("1/x", 210, 138, 36, 29)

Local $idBtnSqrt = GUICtrlCreateButton("Sqrt", 210, 73, 36, 29)

Local $idBtnPercentage = GUICtrlCreateButton("%", 210, 106, 36, 29)

Local $idBtnBackspace = GUICtrlCreateButton("Backspace", 54, 37, 63, 29)

Local $idBtnClearE = GUICtrlCreateButton("CE", 120, 37, 62, 29)

Local $idBtnClear = GUICtrlCreateButton("C", 185, 37, 62, 29)

Local $idEdtScreen = GUICtrlCreateEdit("0.", 8, 2, 239, 23)
Local $idLblMemory = GUICtrlCreateLabel("", 12, 39, 27, 26)

GUISetState()

Local $msg

Do

    $msg = GUIGetMsg()

Until $msg = $GUI_EVENT_CLOSE

Local $idEdtScreen = GUICtrlCreateEdit("0.", 8, 2, 239, 23, BitOR($ES_READONLY,
$ES_RIGHT), $WS_EX_STATICEDGE)

Local $idLblMemory = GUICtrlCreateLabel("", 12, 39, 27, 26, $SS_SUNKEN)
```

Operations:

```
$1 = InputBox("Maths", "Number:", "")

$operation = InputBox("Maths", "+, -, *:", "")

$2 = InputBox("Maths", "plus Number:", "")

$plus = $1 + $2

$minus = $1 - $2

$times = $1 * $2

If $operation = "+" Then

        MsgBox("", "Maths", "= " & $plus)

EndIf

If $operation = "-" Then

        MsgBox("", "Maths", "= " & $minus)

EndIf

If $operation = "*" Then

        MsgBox("", "Maths", "= " & $times)

EndIf
```

Automate:

```
;Launch calc
Run("Calc.exe","")
;wait for 2 seconds
sleep(2000)
If WinExists("Calculator","")Then
   ;Press Button 5
   ControlClick("Calculator","",135)
   sleep(2000)
   ;Press Button +
   ControlClick("Calculator","",93)
   sleep(2000)
   ;Press Button 6
   ControlClick("Calculator","",136)
   sleep(2000)
   ;Press Button =
   ControlClick("Calculator","",121)
   sleep(2000)
   ;Close Calc
   WinClose("Calculator","")
   EndIf
```

Conclusion: Thus we have studied and Executed a Program in Autoit v3 to Create Calculator.

Experiment No 8

Aim: Automate Notepad Application using AutoIT.

Theory:

First step we need to know is the name of the Notepad executable. It is notepad.exe - you can get this information by looking at the properties of the Notepad shortcut icon in the Start Menu.

To execute Notepad we use the AutoIt Run function. This function simply launches a given executable and then continues.

Type in the first line of script as:

Run("notepad.exe")

Run the script - if all goes well then a new instance of Notepad should open. When automating applications AutoIt can check for window title so that it knows which window it should work with. With Notepad the window title is obviously **Untitled - Notepad**.

AutoIt is case-sensitive when using window titles so you must get the title exactly right - the best way to do this is to use the AutoIt Window AutoIt v3 Window Info. Run the AutoIt v3 Window Info from **Start Menu \ AutoIt v3 \ AutoIt Window Info**.

With the Info Tool open click on the newly opened Notepad window to activate it; the Info Tool will give you information about it. The information we are interested in is the window **title**.

Enter the following as the second line in the script (use **CTRL-V** or **Edit\Paste** to paste our window title from the clipboard).

WinWaitActive("Untitled - Notepad")

After we are sure the Notepad window is visible we want to type in some text. This is done with the Send function.

Add this line to our script.

Send("This is some text.")

Next we want to close notepad, we can do this with the WinClose function.

WinClose("Untitled - Notepad")

So, we add a line to wait for this dialog to become active (we will also use the window text to make the function more reliable and to distinguish this new window from the original Notepad window):

WinWaitActive("Notepad", "Save")

Next we want to automatically press Alt-N to select the No/Don't save button (Underlined letters in windows usually indicate that you can use the ALT key and that letter as a keyboard shortcut). In the Send function to send an ALT key we use ! .

```
Send("!n")
```

Our complete script now looks like this:

```
Run("notepad.exe")
WinWaitActive("Untitled - Notepad")
Send("This is some text.")
WinClose("Untitled - Notepad")
WinWaitActive("Notepad", "Save")
;WinWaitActive("Notepad", "Do you want to save") ; When running
under Windows XP
Send("!n")
```

Conclusion: We learned techniques in this experiment to automate Windows Notepad application.

Experiment No 9

Aim: Automate any installation procedure- WinZip.

Theory:

1. WinZip installation consists of approximately 10 dialog boxes that you must click buttons (usually Next) to continue. We are going to write a script that simply waits for these dialog boxes to appear and then clicks the appropriate buttons.

2. As usual with these types of installations the window title of each dialog is the same (WinZip Setup) so we must use window text to tell the difference between windows.

3. First create a directory that we will use for the WinZip installer and our script file. Copy the WinZip installer to this directory and create a blank script called **winzipinstall.au3**.

4. We will now run through the installation manually and write the script as we go. The script lines to automate each dialog will be shown after each picture

5. The first script line is easy, we want to run the **winzip90.exe** installer. So the first line is:

```
Run("winzip90.exe")
```

6. We need to wait for this window to popup and become active, and then we need to send the keystroke ALT-s to click the Setup button. The resulting script lines are:

```
WinWaitActive("WinZip®9.0SR-1Setup", "&Setup")
Send("!s")
```

7. We need to wait for this screen to be active and then click ENTER to accept the install location. The script lines are:

```
WinWaitActive("WinZipSetup", "into the following folder")
Send("{ENTER}")
```

8. Notice that this window has exactly the same title as the first of WinZip Setup - in fact all the dialogs in the setup have this title! In order to tell the difference between these windows we must also use the window text - on each screen try to pick the most unique text you can. In this case I've chosen WinZip features include. After the window has appeared we will want to press ALT-n:

```
WinWaitActive("WinZip Setup", "WinZip features include")
Send("!n")
```

Wait for the window to appear and then press **ALT-y** to accept the agreement:

```
WinWaitActive("License Agreement")
Send("!y")
```

9. Setup continues in a similar fashion for a few dialogs. The picture of each dialog is shown along with the script lines needed to automate it.

```
WinWaitActive("WinZip Setup", "Quick Start Guide")
Send("!n")
```

```
WinWaitActive("WinZip Setup", "switch between the two interfaces")
Send("!c")
Send("!n")
```

```
WinWaitActive("WinZip Setup", "&Express setup (recommended)")
Send("!e")
Send("!n")
```

```
WinWaitActive("WinZip Setup", "WinZip needs to associate itself with your archives")
Send("!n")
```

10. This is the final dialog of the setup. Notice that the Finish button doesn't have a keyboard shortcut - luckily it is the default button on this dialog so we can just press ENTER to select it. If this were not the case then we would have to TAB between controls or better yet use the ControlClick function.

```
WinWaitActive("WinZip Setup", "Thank you for installing this evaluation version")
Send("{ENTER}")
```

```
WinWaitActive("WinZip (Evaluation Version)")
WinClose("WinZip (Evaluation Version)")
```

Experiment No 10

Aim: Study and Demonstration of Bug Tracking Tool -. Bugzilla

Theory:

Bugzilla is an open-source tool used for issues and bugs tracking system. It is widely used as a bug-reporting tool for all types of testing functions.

Bugzilla is an open-source tool used to track bugs and issues of a project or a software. It helps the developers and other stakeholders to keep track of outstanding problems with the product.

- It was written by **Terry Weissman** in TCL programming language in 1998.
- Later, Bugzilla was written in PERL and it uses the MYSQL database.

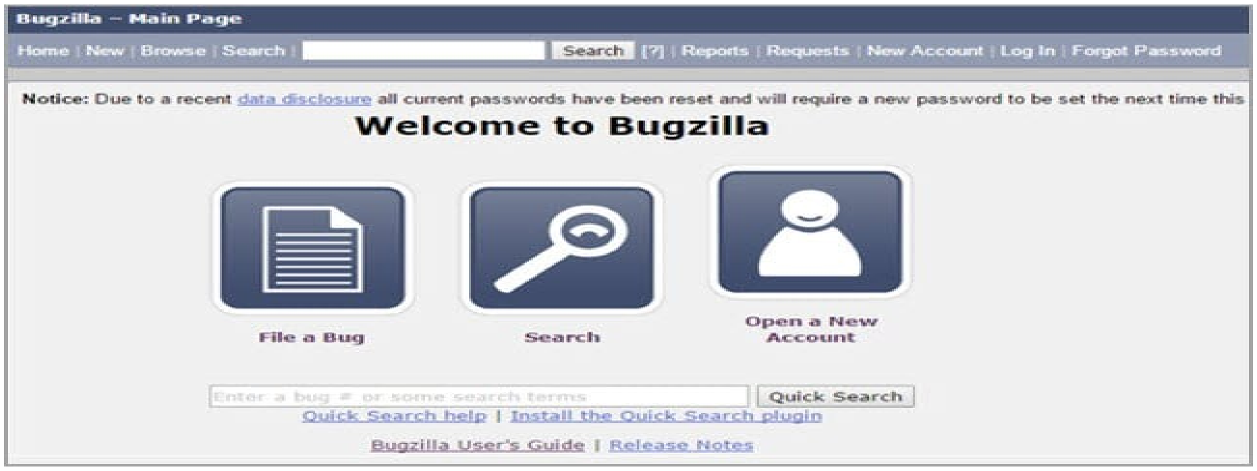
Software Requirements:

- Perl
- Database Server
- Web Server

Bugzilla. Download and install

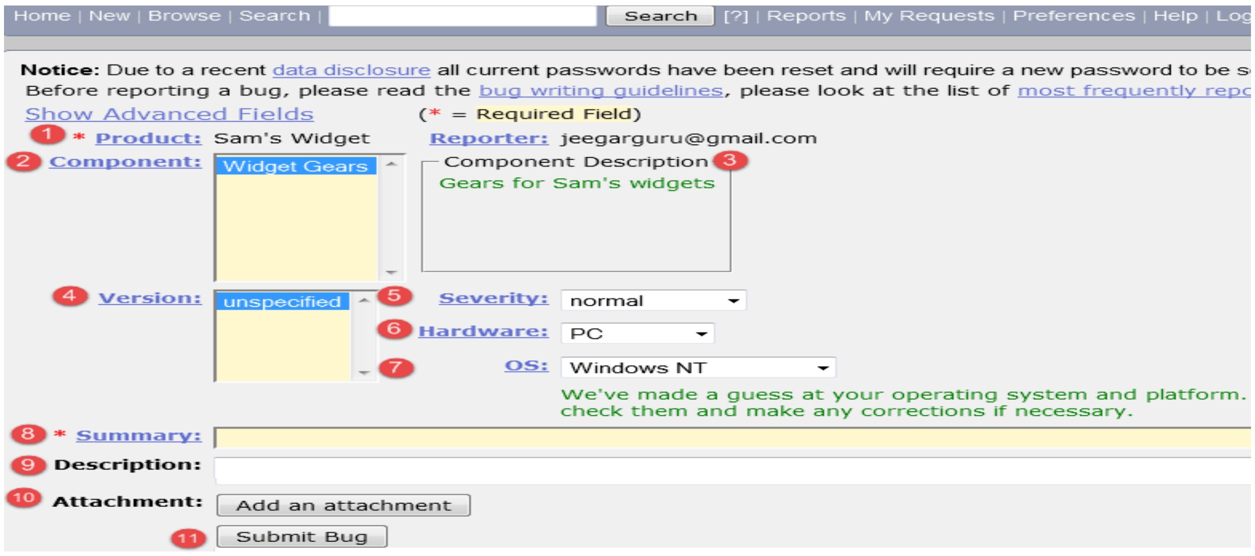
git clone --branch release-X.X-stable https://github.com/bugzilla/bugzilla

1. Log-in


Bugzilla – Main Page
Home | New | Browse | Search | Search [?] | Reports | Requests | New Account | Log In | Forgot Password
Welcome to Bugzilla
File a Bug
Search
Open a New Account
Enter a bug # or some search terms
Quick Search
Quick Search help | Install the Quick Search plugin
Bugzilla User's Guide | Release Notes

Notice: Due to a recent data disclosure all current passwords have been reset and will require a new password to be set the next time this

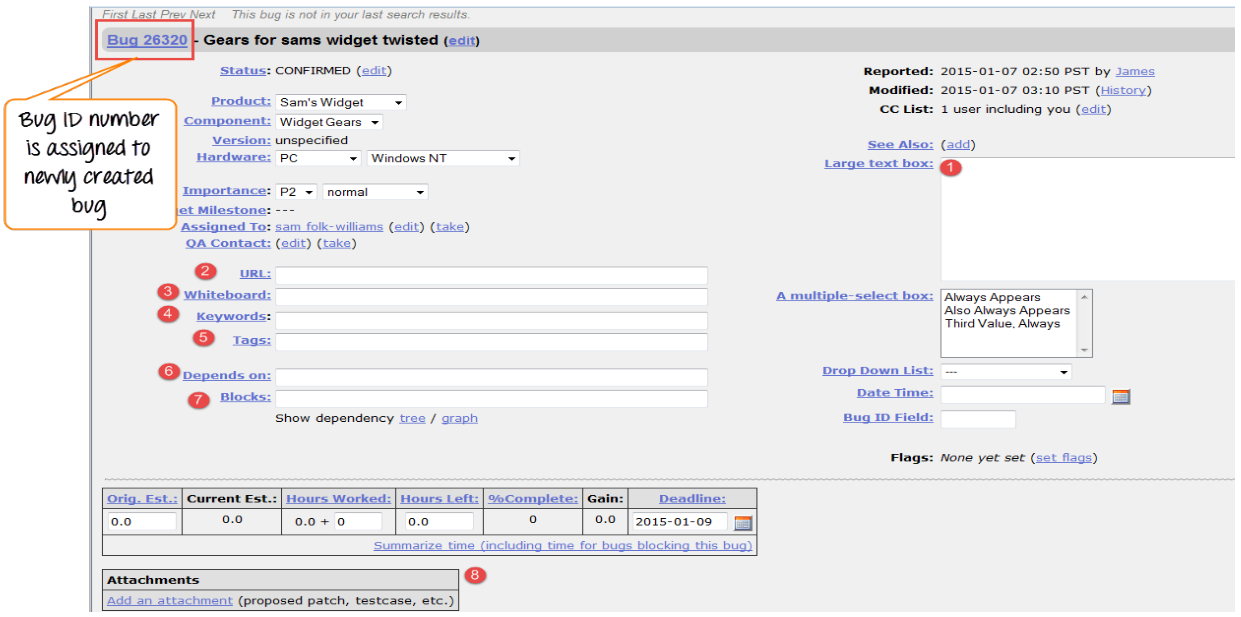
2) In the next window


Home | New | Browse | Search | Search | [?] | Reports | My Requests | Preferences | Help | Log
1
2
3
4
5
6
7
8
9
10
11

Notice: Due to a recent data disclosure all current passwords have been reset and will require a new password to be s
Before reporting a bug, please read the bug writing guidelines, please look at the list of most frequently repo
Show Advanced Fields (* = Required Field)

* **Product:**	Sam's Widget
Reporter:	jeegarguru@gmail.com
Component:	Widget Gears
Component Description	Gears for Sam's widgets
Version:	unspecified
Severity:	normal
Hardware:	PC
OS:	Windows NT
	We've made a guess at your operating system and platform. check them and make any corrections if necessary.
* **Summary:**	
Description:	
Attachment:	Add an attachment
	Submit Bug

3) Bug is created



First Last Prev Next This bug is not in your last search results.

Bug 26320 - Gears for sams widget twisted (edit)

Status:	CONFIRMED (edit)
Product:	Sam's Widget
Component:	Widget Gears
Version:	unspecified
Hardware:	PC Windows NT
Importance:	P2 normal
et Milestone:	---
Assigned To:	sam folk-williams (edit) (take)
QA Contact:	(edit) (take)
URL:	
Whiteboard:	
Keywords:	
Tags:	
Depends on:	
Blocks:	
	Show dependency tree / graph

Reported:	2015-01-07 02:50 PST by James
Modified:	2015-01-07 03:10 PST (History)
CC List:	1 user including you (edit)
See Also:	(add)
Large text box:	
A multiple-select box:	Always Appears Also Always Appears Third Value, Always
Drop Down List:	---
Date Time:	
Bug ID Field:	
Flags:	*None yet set* (set flags)

Orig. Est.:	Current Est.:	Hours Worked:	Hours Left:	%Complete:	Gain:	Deadline:
0.0	0.0	0.0 + 0	0.0	0	0.0	2015-01-09

Summarize time (including time for bugs blocking this bug)

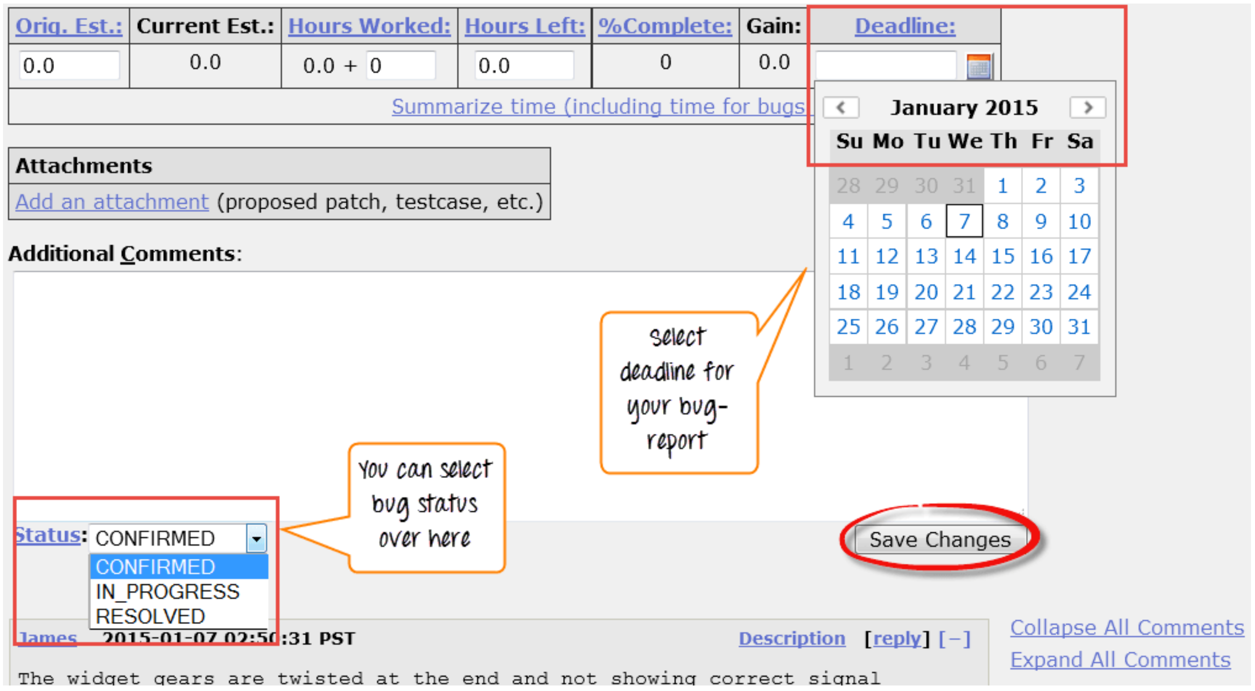
Attachments
Add an attachment (proposed patch, testcase, etc.)

The fields that are present in the default bug screen are:

- **Product –** Which we selected on the previous page
- **Component –** Each product can be split into one or more components based on the use or functionality etc.
- **Version –** Version of the product in which the bug was detected
- **Reporter –** Email id of the person logging the bug

- **Severity –** Severity of the bug
- **Hardware and OS –** Machine details from which bug is logged
- **Summary –** To provide summary for the bug
- **Description –** A complete description of the bug
- **Add an Attachment –** To provide any supporting file as attachment
- **Submit a Bug –** To submit the bug and create a Bug ID

4) Deadline in Bugzilla usually gives the time-limit to resolve the bug in given time frame.



Orig. Est.:	Current Est.:	Hours Worked:	Hours Left:	%Complete:	Gain:	Deadline:
0.0	0.0	0.0 + 0	0.0	0	0.0	

Summarize time (including time for bugs

January 2015						
Su	Mo	Tu	We	Th	Fr	Sa
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Attachments
Add an attachment (proposed patch, testcase, etc.)

Additional Comments:

Status: CONFIRMED

James 2015-01-07 02:50:31 PST Description [reply] [−]

```
The widget gears are twisted at the end and not showing correct signal
```

Conclusion: Thus we have studied Bug Tracking Tool -. Bugzilla.

Experiment No 11

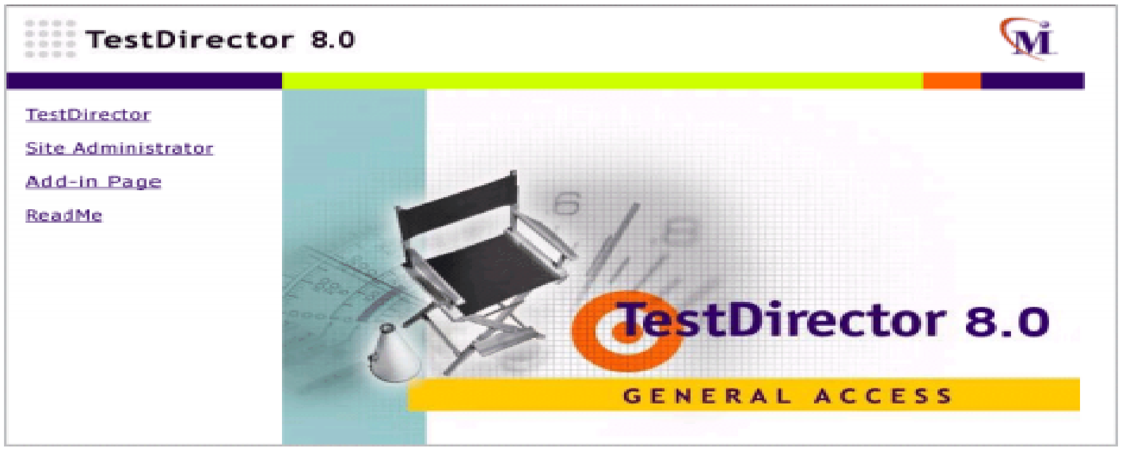
Aim: Study and Demonstration of test management Tool -Test Director

Theory:

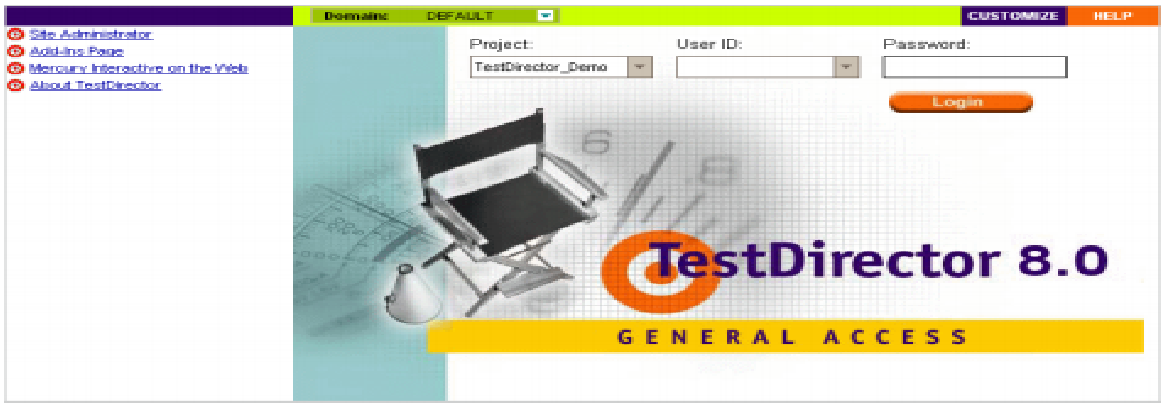
TestDirector offers an organized framework for testing applications before they are deployed. Since test plans evolve with new or modified application requirements, you need a central data repository for organizing and managing the testing process. TestDirector guides you through the requirements specification, test planning, test execution, and defect tracking phases of the testing process. The TestDirector testing process includes four phases:

1 Open the TestDirector Options window.

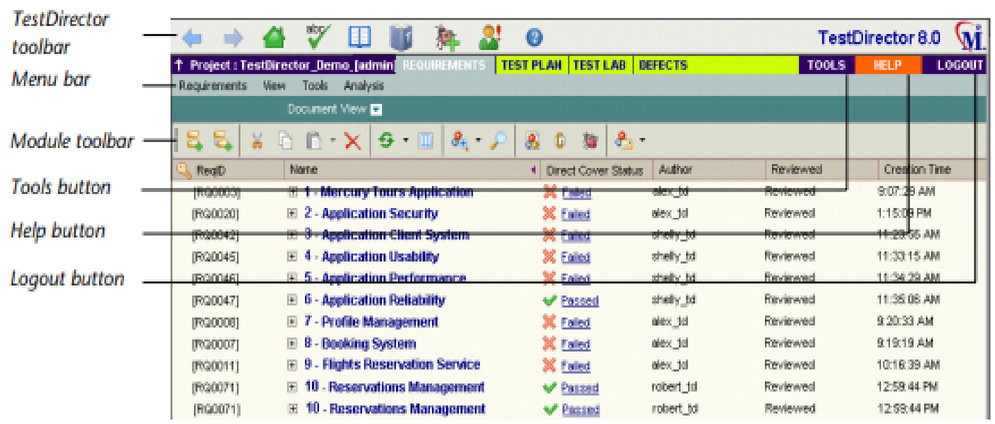
http:/localhost:81/virtuldirectory1//default.htm



2. The Test Director Login window opens.



3. Explore the common TestDirector elements.



ReqID	Name	Direct Cover Status	Author	Reviewed	Creation Time
[RQ0003]	1 - Mercury Tours Application	Failed	alex_td	Reviewed	9:07:28 AM
[RQ0020]	2 - Application Security	Failed	alex_td	Reviewed	1:15:09 PM
[RQ0042]	3 - Application Client System	Failed	shelly_td	Reviewed	11:28:56 AM
[RQ0045]	4 - Application Usability	Failed	shelly_td	Reviewed	11:33:15 AM
[RQ0046]	5 - Application Performance	Failed	shelly_td	Reviewed	11:34:29 AM
[RQ0047]	6 - Application Reliability	Passed	shelly_td	Reviewed	11:35:06 AM
[RQ0008]	7 - Profile Management	Failed	alex_td	Reviewed	9:20:33 AM
[RQ0007]	8 - Booking System	Failed	alex_td	Reviewed	9:19:19 AM
[RQ0011]	9 - Flights Reservation Service	Failed	alex_td	Reviewed	10:16:39 AM
[RQ0071]	10 - Reservations Management	Passed	robert_td	Reviewed	12:59:44 PM
[RQ0071]	10 - Reservations Management	Passed	robert_td	Reviewed	12:59:44 PM

1 Analyze your application and determine your testing requirements.

To specify your testing requirements,

Consider the following steps: ➤ Examine the application documentation to determine your testing scope—test goals, objectives, and strategies.

Create a test plan, based on your testing requirements.

2. To create a test plan,

Consider the following steps: ➤ Examine your application, system environment, and testing resources to determine your testing goals. ➤ Divide your application into modules or functions to be tested. Build a test plan tree to hierarchically divide your application into testing units, or subjects

3. Submit defects detected in your application and track the progress of defect fixes.

Submit new defects detected in your application. Quality assurance testers, developers, project managers, and end users can add defects during any phase of the testing process.

Conclusion: Thus we have studied Test Management Tool -. TestDirector