

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING



SOFTWARE TESTING METHODOLOGIES

LAB MANUAL

Subject Code : **CS625PE**

Regulation : **R18/JNTUH**

Academic Year : **2020-2021**

III B. TECH II SEMESTER

COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated to JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION AND MISSION OF THE INSTITUTION

VISION

To become self-sustainable institution this is recognized for its new age engineering through innovative teaching and learning culture, inculcating research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION

- To offer undergraduate and post-graduate programs that is supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
- To provide necessary support structures for students, this will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
- To provide faculty and students with an ecosystem that fosters research and development through strategic partnerships with government organisations and collaboration with industries.
- To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic growth.

VISION AND MISSION OF CSE DEPARTMENT

VISION

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION

- To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills while working with the community to address societal issues.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES (PEOS):

A graduate of the Computer Science and Engineering Program should:

PEO1	Program Educational Objective1: (PEO1) The Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.
PEO2	Program Educational Objective2 :(PEO2) The Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.
PEO3	Program Educational Objective3: (PEO3) The Graduates will communicate effectively, work collaboratively and exhibit high levels of Professionalism, moral and ethical responsibility.
PEO4	Program Educational Objective4 :(PEO4) The Graduates will develop the ability to understand and analyse Engineering issues in a broader perspective with ethical responsibility towards sustainable development.

PROGRAM OUTCOMES (POS):

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering Fundamentals and an engineering specialization to the solution of complex engineering problems.
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and sustainability: Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	Individual and team work: Function effectively as an individual, and as a member or leader In diverse teams, and in multi-disciplinary settings.
PO10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	Project management and finance: Demonstrate knowledge and understanding of the Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES(PSOS):

PSO1	Problem Solving Skills – Graduate will be able to apply computational techniques and software principles to solve complex engineering problems pertaining to software engineering.
PSO2	Professional Skills – Graduate will be able to think critically, communicate effectively, and collaborate in teams through participation in co and extra-curricular activities.
PSO3	Successful Career – Graduates will possess a solid foundation in computer science and engineering that will enable them to grow in their profession and pursue lifelong learning through post-graduation and professional development.



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PREREQUISITES:

A basic knowledge of programming.

COURSE OBJECTIVES:

1. To provide knowledge of Software Testing Methods.
2. To develop skills in software test automation and management using latest tools.

COURSE OUTCOME

1. Design and develop the best test strategies in accordance to the development model

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Name: STM LAB

Course Code: CS625PE

Year/Semester: III/II

Regulation: R18

S. No	List of Experiments	Page No.
1	Recording in context sensitive mode and analog mode	
2	GUI checkpoint for single property	
3	GUI checkpoint for single object/window	
4	GUI checkpoint for multiple objects	
5	a) Bitmap checkpoint for object/window a) Bitmap checkpoint for screen area	
6	Database checkpoint for Default check	
7	Database checkpoint for custom check	
8	Database checkpoint for runtime record check	
9	a) Data driven test for dynamic test data submission b) Data driven test through flat files c) Data driven test through front grids d) Data driven test through excel test	
10	a) Batch testing without parameter passing b) Batch testing with parameter passing	
11	Data driven batch	
12	Silent mode test execution without any interruption	
13	Test Case For Calculator In Windows Application	
14	Test Cases For Mobile Application Testing	
15	Test Cases For Cloud Environment Testing	
16	Test Cases for Pen	

FACULTY

HOD, CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INTRODUCTION TO SOFTWARE TESTING

Software Testing is the process of identifying the accuracy and quality of the software product and service under test. Apparently, it was born to validate whether the product fulfils the particular prerequisites, needs, and desires of the client. At the end of the day, testing executes a framework or application with a specific end goal to point out bugs, errors or defects. The responsibility of testing is to point out the issues of bugs and give Dev (Developers) a clue to help them fix it right following the requirements.

1. Software Testing Objectives:

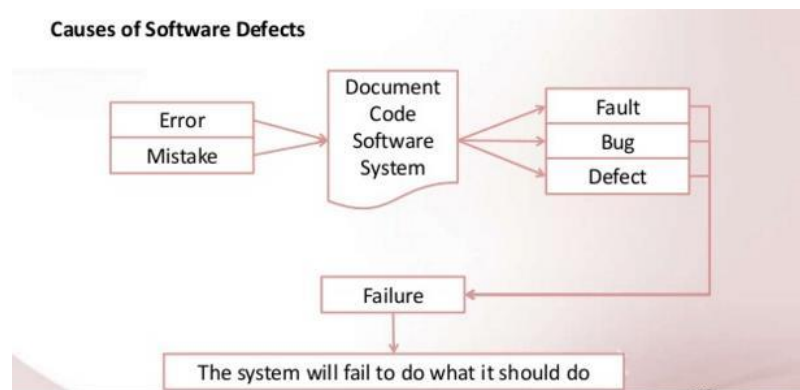
- Uncover as many as errors (or bugs) as possible in a given product.
- Demonstrate a given software product matching its requirement specifications.
- Validate the quality of a software using the minimum cost and efforts.
- Generate high-quality test cases, perform effective tests, and issue correct and helpful problem reports.

Software testing is often divided into 2 main processes: Verification and Validation.

Verification in software testing is the process when your team just need to check whether the software, system or framework consistent, aligned with the requirements of a documentation.

Validation is the process that your team needs to verify accuracy of the system. In this process, you will look back to product, system and think about what users actually want and what has been done.

In Software Testing, there is the difference between Errors, Defects, and Bugs that we should distinguish clearly to avoid misunderstanding problem.



Error is a deviation from the actual and the expected result. It represents the mistakes made by people.

Bug is an error found BEFORE the application goes into production. A programming error that causes a program to work poorly, produce incorrect results, or crash. An error in software or hardware that causes a program to malfunction.

Defect happens once the error is identified during testing; it is logged as a 'Defect' in the tracking system.

Failure is the incapacity of a system to conduct its required functions within clarified performance requirements, literally a disappointment or a letdown. And no one wants to do business with a failure.

2. Why is Software Testing important?

One of the controversy topics that is mentioned in a long meeting before starting or after reviewing a project's sprint is about the associated costs of fixing bugs. The more extended a bug goes undetected, the more exorbitant it is to settle. Basic costs versus benefits investigation will indicate overwhelmingly that the advantages of utilizing a QA Testing Engineer to approve the code far exceed the expenses.

Besides the expensive cost needed to be paid for the delay of detecting bugs, if the testing team missed out on capturing or identifying the risks and software issues accurately or exhaustively it can lead to some disasters. For example: It took NASA 7 years to identify gigantic hole in the sky due to an error in the data analysis software, the huge hole in the ozone was flagged as a software issue in the system and was ignored because they did not expect such an extreme deviation in the results thrown up by the software.

When NASA engineers reviewed their raw data, they realized that their satellites had indeed detected the hole long ago, but they completely missed it. So that, here are 5 reasons why every member of the team should take an eye on Product Testing for the whole process.

To ensure there is no difference between the actual and the expectation

Software testing is created to detect any problems occurred during coding and developing a new function or feature of a software product before it is delivered to the end users. These upcoming features should be correctly matched with what it is supposed to be. Also, Testing tool will be a

gadget checking whether your team decently refers to the requirement. Hence, it basically is a device helping to close the gap between actual and expectation in making a software product.

To make sure your product is powerful enough no matter how many people are using.

There is a big distinction when there is a person using your product referring to hundreds of people trying to do the same thing at the same time. Your software needs to be strong enough to guarantee that there will be no crashing down or loading annoying happened when a number of people are trying to run your products. Therefore, it should be smoothly and perfectly working with everyone.

To figure out as many as possible the chance of bugs generating.

It can't be denied to say that nothing is perfect. There is always some unseen problems which may generate during using your application. The responsibility of testing tool is to avoid bugs found by users. We, who develop this application/software product should take the duty of falling down as many as possible the number of bugs which may interrupt users in future, so deliver the best experience for our users whilst using our apps.

To offer a product that can perfectly work on different browsers and tech devices

At the booming moment of this technology era, we, fortunately, welcome the existence of a number of technology devices, browsers, and operating systems, giving us the chance to choose different technology instruments to greater our tech experience. Therefore, the stress of creating an application or product which could perfectly work on most of the technology instruments has never been so great before.

To deliver the best product that we can.

Again, testing tool is created to provide the most excellent software product and service to the end users. Of course, we couldn't bring the best (since there is nothing perfect, we all know) but we could minimize the chance of bugs occurred within our capability. Before releasing, we could be proud and confident enough on the product we bring to market. In any case, unseen bugs may have a real impact on a real person. Therefore, if we could have a chance to encounter the bug before the users find it out, nothing could be greater than this.

3. Type of Software Testing:

Testing is surely a fundamental part of software development. Poor testing methodologies cause the troublesome products and unsustainable development. Having a well-prepared testing plan makes a product be more competitive and assure the products coming in a predictable timeline associated with high quality.

Apparently, a product is usually tested from a very early stage when it is just a small code tested piece by piece then being tested at the final of development when it is under the shape of a full application or software product in general. Of course, there are a number of Software Testing types out there (more than 100 different types in general); however, at the beginning, we just need to adjust a few common types that every product usually goes through before going further.

Unit Test

It is not exaggerated saying that people usually hear about Unit Test before getting noted about the software testing industry since it is the most basic testing used at the developer level. We focus on the single piece of unit code whilst it is already isolated from any outside interaction or dependency on any module before. This test requires the developer checking the smallest units of codes they have written and prove that units can work independently.

Integration Test

Still, at the developer level, after Unit Test, the combination (or integration) of these smallest codes should also be checked carefully. Integration test provides the testing modules which access to network, databases and file system. They will indicate whether the database and the network are working well when they are combined into the whole system. Most importantly, the connection between small units of code tested in the previous stage will be proven at this stage.

Functional Testing

There is no doubt to claim that functional testing is the higher level of test type should be used after Integration Test. Functional tests check for the accuracy of the output with respect to the input defined in the specification. Not much emphasis is given to the intermediate values but more focus is given on the final output created.

Smoke Test

Smoke Tests analogy comes from the electronics where an issue means the circuit board giving out smoke. After functional tests are done, a simple test will be executed at the starting point, after a fresh installation and newer input values.

Regression Test

Whenever complex bugs are stuck in a system, typically which affect the core areas of the system, regression tests are used to retest all the modules of the system.

UI Test

Well, apart from those core testing types above, GUI test is also a well-known and really popular in software engineering industry now. This graphic user interface testing ensures the specific application or products being friendly for all users. Principally, GUI Testing evaluates design components such as layout, colors, fonts, size, and so on. Also, GUI Testing can be executed both manually and automatically.

This is only a brief introduction to what is software testing. If you are interested in learning more about the discipline, you might want to start with webinars and eBooks on Huddle or try this post on Manual Testing & Automated Testing.

EXPERIMENT 1

RECORDING IN CONTEXT SENSITIVE MODE AND ANALOG MODE

Context Sensitive mode records the operations you perform on your application in terms of its GUI objects. As you record, WinRunner identifies each GUI object you click (such as a window, button, or list), and the type of operation performed (such as drag, click, or select).

For example, if you click the **Open** button in an Open dialog box, WinRunner records the following:
`button_press ("Open");`

When it runs the test, WinRunner looks for the Open dialog box and the Open button represented in the test script. If, in subsequent runs of the test, the button is in a different location in the Open dialog box, WinRunner is still able to find it.



In version 1, the Open button is above the Cancel button.

In version 2, the Open button is below the Cancel button.



Use Context Sensitive mode to test your application by operating on its user interface.

To record a test in context sensitive mode:

1. Choose **Test > Record–Context Sensitive** or click the **Record–Context Sensitive** button.

The letters **Rec** are displayed in dark blue text with a light blue background on the **Record** button to indicate that a context sensitive record session is active.

2. Perform the test as planned using the keyboard and mouse.

Insert checkpoints and synchronization points as needed by choosing the appropriate commands from the User toolbar or from the **Insert** menu menu: GUI Checkpoint, Bitmap Checkpoint, Database Checkpoint, or Synchronization Point.

3. To stop recording, click **Test > Stop Recording** or click **Stop**.

EXPERIMENT 2

GUI CHECKPOINT FOR SINGLE PROPERTY

You can check a single property of a GUI object. For example, you can check whether a button is enabled or disabled or whether an item in a list is selected. To create a GUI checkpoint for a property value, use the Check Property dialog box to add one of the following functions to the test script:

button_check_info scroll_check_info

edit_check_info static_check_info

list_check_info win_check_info

obj_check_info

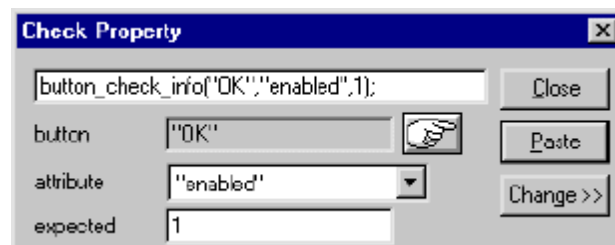
To create a GUI checkpoint for a property value:

1. Choose **Insert > GUI Checkpoint > For Single Property**. If you are recording in Analog mode, press the CHECK GUI FOR SINGLE PROPERTY softkey in order to avoid extraneous mouse movements.

The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.

2. Click an object.

The Check Property dialog box opens and shows the default function for the selected object. WinRunner automatically assigns argument values to the function.



1. You can modify the arguments for the property check.
 - To modify assigned argument values, choose a value from the **Attribute** list. The expected value is updated in the Expected text box.
 - To choose a different object, click the pointing hand and then click an object in your application. WinRunner automatically assigns new argument values to the function.

Note: that if you click an object that is not compatible with the selected function, a message states that the current function cannot be applied to the selected object. Click OK to clear the message, and then click Close to close the Check Property dialog box. Repeat steps 1 and 2.

2. Click **Paste** to paste the statement into your test script.

EXPERIMENT 3

GUI CHECKPOINT FOR SINGLE OBJECT/WINDOW

You can create a GUI checkpoint to check a single object in the application being tested. You can either check the object with its default properties or you can specify which properties to check.

Each standard object class has a set of default checks. For a complete list of standard objects, the properties you can check, and default checks, see “Property Checks and Default Checks”.

Creating a GUI Checkpoint using the Default Checks

You can create a GUI checkpoint that performs a default check on the property recommended by WinRunner. For example, if you create a GUI checkpoint that checks a push button, the default check verifies that the push button is enabled.

To create a GUI checkpoint using default checks:

1. Choose **Insert > GUI Checkpoint >for Object/Window**, or click the **GUI Checkpoint for Object/Window** button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR OBJECT/WINDOW soft key in order to avoid extraneous mouse movements. Note that you can press the CHECK GUI FOR OBJECT/WINDOW soft key in Context Sensitive mode as well.

The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.

2. Click an object.
3. WinRunner captures the current value of the property of the GUI object being checked and stores it in the test's expected results folder. The WinRunner window is restored and a GUI checkpoint is inserted in the test script as an **obj_check_gui** statement.

Creating a GUI Checkpoint by Specifying which Properties to Check

You can specify which properties to check for an object. For example, if you create a checkpoint that checks a push button, you can choose to verify that it is in focus, instead of enabled.

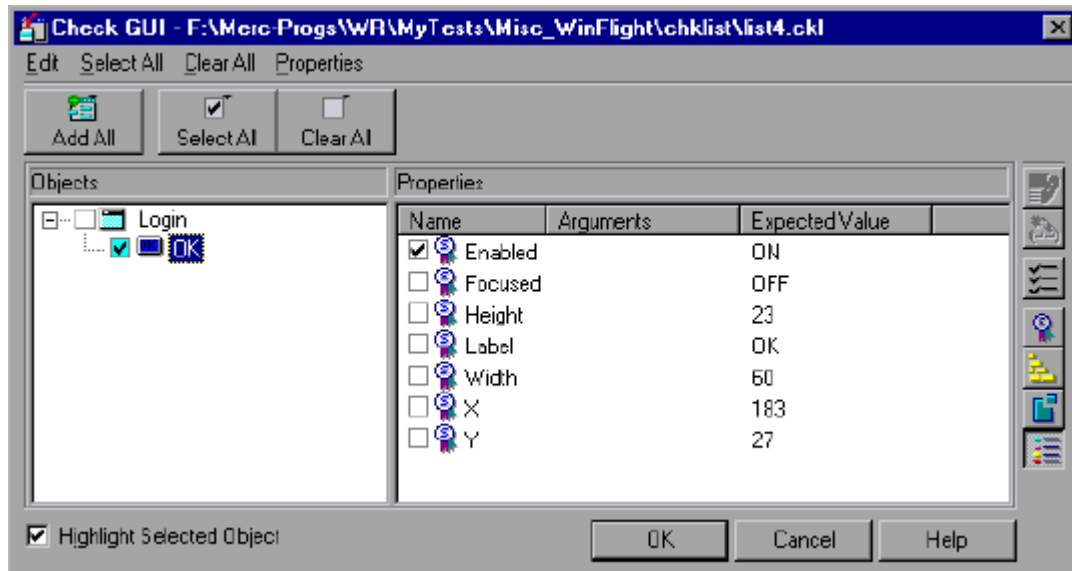
To create a GUI checkpoint by specifying which properties to check:

1. Choose **Insert > GUI Checkpoint >for Object/Window**, or click the **GUI Checkpoint for Object/Window** button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR OBJECT/WINDOW softkey in order to avoid extraneous mouse movements.

Note that you can press the CHECK GUI FOR OBJECT/WINDOW soft key in Context Sensitive mode as well.

The Win Runner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens on the screen.

2. Double-click the object or window. The Check GUI dialog box opens.



1. Click an object name in the **Objects** pane. The **Properties** pane lists all the properties for the selected object.
2. Select the properties you want to check.
 - To edit the expected value of a property, first select it. Next, either click the **Edit Expected Value** button, or double-click the value in the **Expected Value** column to edit it.
 - To add a check in which you specify arguments, first select the property for which you want to specify arguments. Next, either click the **Specify Arguments** button, or double-click in the **Arguments** column. Note that if an ellipsis (three dots) appears in the Arguments column, then you must specify arguments for a check on this property. (You do not need to specify arguments if a default argument are specified.) When checking standard objects, you only specify arguments for certain properties of edit and static text objects. You also specify arguments for checks on certain properties of nonstandard objects.
 - To change the viewing options for the properties of an object, use the **Show Properties** buttons.
3. Click **OK** to close the Check GUI dialog box.



Win Runner captures the GUI information and stores it in the test's expected results folder. The Win Runner window is restored and a GUI checkpoint is inserted in the test script as an **obj_check_gui** or a **win_check_gui** statement. For more information, see "Understanding GUI Checkpoint Statements".

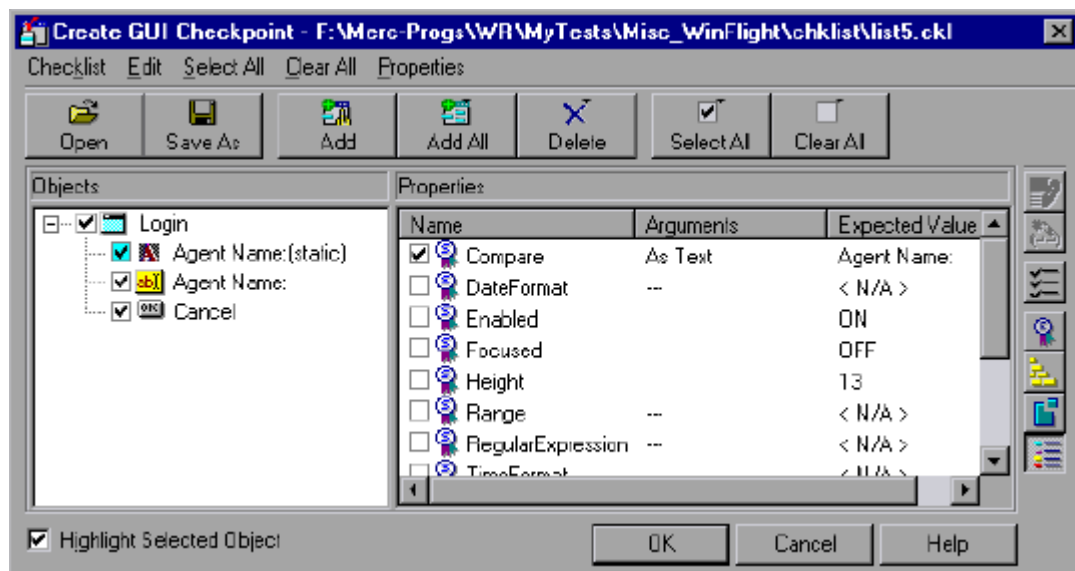
EXPERIMENT 4

GUI checkpoint for multiple objects

You can use a GUI checkpoint to check two or more objects in a window. For a complete list of standard objects and the properties you can check, see “Property Checks and Default Checks”.

To create a GUI checkpoint for two or more objects:

1. Choose **Insert > GUI Checkpoint > For Multiple Objects** or click the **GUI Checkpoint for Multiple Objects** button on the User toolbar. If you are recording in Analog mode, press the CHECK GUI FOR MULTIPLE OBJECTS softkey in order to avoid extraneous mouse movements. The Create GUI Checkpoint dialog box opens.
2. Click the **Add** button. The mouse pointer becomes a pointing hand and a help window opens.
3. To add an object, click it once. If you click a window title bar or menu bar, a help window prompts you to check all the objects in the window.
4. The pointing hand remains active. You can continue to choose objects by repeating step 3 above for each object you want to check.
5. Click the right mouse button to stop the selection process and to restore the mouse pointer to its original shape. The Create GUI Checkpoint dialog box reopens.



6. The Objects pane contains the name of the window and objects included in the GUI checkpoint. To specify which objects to check, click an object name in the **Objects** pane. The Properties pane lists all the properties of the object. The default properties are selected.

- To edit the expected value of a property, first select it. Next, either click the **Edit Expected Value** button, or double-click the value in the **Expected Value** column to edit it.
 - To add a check in which you specify arguments, first select the property for which you want to specify arguments. Next, either click the **Specify Arguments** button, or double-click in the **Arguments** column. Note that if an ellipsis appears in the Arguments column, then you must specify arguments for a check on this property. (You do not need to specify arguments if a default argument is specified.) When checking standard objects, you only specify arguments for certain properties of edit and static text objects. You also specify arguments for checks on certain properties of nonstandard objects.
 - To change the viewing options for the properties of an object, use the Show Properties buttons.
7. To save the checklist and close the Create GUI Checkpoint dialog box, click **OK**.

WinRunner captures the current property values of the selected GUI objects and stores it in the expected results folder. A **win_check_gui** statement is inserted in the test script.

EXPERIMENT 5

A. Bitmap checkpoint for object/window

You can capture a bitmap of any window or object in your application by pointing to it. The method for capturing objects and for capturing windows is identical. You start by choosing **Insert > Bitmap Checkpoint > For Object/Window**. As you pass the mouse pointer over the windows of your application, objects and windows flash. To capture a window bitmap, you click the window's title bar. To capture an object within a window as a bitmap, you click the object itself.

Note that during recording, when you capture an object in a window that is not the active window, WinRunner automatically generates a **set_window** statement.

To capture a window or object as a bitmap:

- Choose **Insert > Bitmap Checkpoint > For Object/Window** or click the **Bitmap Checkpoint for Object/Window** button on the User toolbar. Alternatively, if you are recording in Analog mode, press the CHECK BITMAP OF OBJECT/WINDOW softkey.

The WinRunner window is minimized, the mouse pointer becomes a pointing hand, and a help window opens.

- Point to the object or window and click it. WinRunner captures the bitmap and generates a **win_check_bitmap** or **obj_check_bitmap** statement in the script.

The TSL statement generated for a window bitmap has the following syntax:

win_check_bitmap (*object, bitmap, time*);

For an object bitmap, the syntax is:

obj_check_bitmap (*object, bitmap, time*);

For example, when you click the title bar of the main window of the Flight Reservation application, the resulting statement might be:

win_check_bitmap ("Flight Reservation", "Img2", 1);

However, if you click the Date of Flight box in the same window, the statement might be:

obj_check_bitmap ("Date of Flight:", "Img1", 1);

B. Bitmap checkpoint for screen area

When working in Context Sensitive mode, you can capture a bitmap of a window, object, or of a specified area of a screen. WinRunner inserts a checkpoint in the test script in the form of either a **win_check_bitmap** or **obj_check_bitmap** statement.

To check a bitmap, you start by choosing **Insert > Bitmap Checkpoint**. To capture a window or another GUI object, you click it with the mouse. To capture an area bitmap, you mark the area to be checked using a crosshairs mouse pointer.

Note that when you record a test in Analog mode, you should press the CHECK BITMAP OF WINDOW softkey or the CHECK BITMAP OF SCREEN AREA softkey to create a bitmap checkpoint. This prevents WinRunner from recording extraneous mouse movements. If you are programming a test, you can also use the Analog function **check_window** to check a bitmap.

If the name of a window or object varies each time you run a test, you can define a regular expression in the GUI Map Editor. This instructs WinRunner to ignore all or part of the name. For more information on using regular expressions in the GUI Map Editor, see “Editing the GUI Map.”

You can include your bitmap checkpoint in a loop. If you run your bitmap checkpoint in a loop, the results for each iteration of the checkpoint are displayed in the test results as separate entries. The results of the checkpoint can be viewed in the Test Results window. For more information, see “Analyzing Test Results.”

EXPERIMENT 6

Database checkpoint for Default check

When you create a default check on a database, you create a standard database checkpoint that checks the entire result set using the following criteria:

- The default check for a multiple-column query on a database is a case sensitive check on the entire result set by column name and row index.
- The default check for a single-column query on a database is a case sensitive check on the entire result set by row position.

If you want to check only part of the contents of a result set, edit the expected value of the contents, or count the number of rows or columns, you should create a custom check instead of a default check. For information on creating a custom check on a database, see “Creating a Custom Check on a Database,”

Creating a Default Check on a Database Using ODBC or Microsoft Query

You can create a default check on a database using ODBC or Microsoft Query.

To create a default check on a database using ODBC or Microsoft Query:

1. Choose **Insert > Database Checkpoint > Default Check** or click the **Default Database Checkpoint** button on the User toolbar. If you are recording in Analog mode, press the CHECK DATABASE (DEFAULT) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (DEFAULT) softkey in Context Sensitive mode as well.

2. If Microsoft Query is installed and you are creating a new query, an instruction screen opens for creating a query.

If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.

Click **OK** to close the instruction screen.

If Microsoft Query is not installed, the Database Checkpoint wizard opens to a screen where you can define the ODBC query manually. For additional information, see “Setting ODBC (Microsoft Query) Options”

3. Define a query, copy a query, or specify an SQL statement. For additional information, see “Creating a Query in ODBC/Microsoft Query” or “Specifying an SQL Statement”

4. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

WinRunner captures the data specified by the query and stores it in the test's exp folder. WinRunner creates the msqr*.sql query file and stores it and the database checklist in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

Creating a Default Check on a Database Using Data Junction

You can create a default check on a database using Data Junction.

To create a default check on a database:

1. Choose **Insert > Database Checkpoint > Default Check** or click the **Default Database Checkpoint** button on the User toolbar.

If you are recording in Analog mode, press the CHECK DATABASE (DEFAULT) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (DEFAULT) softkey in Context Sensitive mode as well.

For information on working with the Database Checkpoint wizard, see "Working with the Database Checkpoint Wizard"

2. An instruction screen opens for creating a query.

If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.

Click **OK** to close the instruction screen.

3. Create a new conversion file or use an existing one. For additional information, see "Creating a Conversion File in Data Junction"
4. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

WinRunner captures the data specified by the query and stores it in the test's exp folder. WinRunner creates the *.djs conversion file and stores it in the checklist in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

EXPERIMENT 7

Database checkpoint for custom check

When you create a custom check on a database, you create a standard database checkpoint in which you can specify which properties to check on a result set.

You can create a custom check on a database in order to:

- check the contents of part or the entire result set
- edit the expected results of the contents of the result set
- count the rows in the result set
- count the columns in the result set

You can create a custom check on a database using ODBC, Microsoft Query or Data Junction.

To create a custom check on a database:

1. Choose **Insert > Database Checkpoint > Custom Check**. If you are recording in Analog mode, press the CHECK DATABASE (CUSTOM) softkey in order to avoid extraneous mouse movements. Note that you can press the CHECK DATABASE (CUSTOM) softkey in Context Sensitive mode as well.
2. Follow the instructions on working with the Database Checkpoint wizard, as described in “Working with the Database Checkpoint Wizard”
3. If you are creating a new query, an instruction screen opens for creating a query.
If you do not want to see this message next time you create a default database checkpoint, clear the **Show this message next time** check box.
4. If you are using ODBC or Microsoft Query, define a query, copy a query, or specify an SQL statement.
If you are using Data Junction, create a new conversion file or use an existing one.
5. If you are using Microsoft Query and you want to be able to parameterize the SQL statement in the **db_check** statement which will be generated, then in the last wizard screen in Microsoft Query, click **View data or edit query in Microsoft Query**. Follow the instructions in “Parameterizing Standard Database Checkpoints”
6. WinRunner takes several seconds to capture the database query and restore the WinRunner window.

The Check Database dialog box opens

EXPERIMENT 8

Database checkpoint for runtime record check

You can add a runtime database record checkpoint to your test in order to compare information displayed in your application during a test run with the current value(s) in the corresponding record(s) in your database. You add runtime database record checkpoints by running the Runtime Record Checkpoint wizard. When you are finished, the wizard inserts the appropriate **db_record_check** statement into your script.

Note that when you create a runtime database record checkpoint, the data in the application and in the database are generally in the same format. If the data is in different formats, you can follow the instructions in “Comparing Data in Different Formats” to create a runtime database record checkpoint. Note that this feature is for advanced WinRunner users only.

Using the Runtime Record Checkpoint Wizard

The Runtime Record Checkpoint wizard guides you through the steps of defining your query, identifying the application controls that contain the information corresponding to the records in your query, and defining the success criteria for your checkpoint.

To open the wizard, select **Insert > Database Checkpoint > Runtime Record Check**.

Define Query Screen

The Define Query screen enables you to select a database and define a query for your checkpoint. You can create a new query from your database using Microsoft Query, or manually define an SQL statement



You can choose from the following options:

- **Create new query:** Opens Microsoft Query, enabling you to create a new query. Once you finish defining your query, you return to WinRunner.
- **Specify SQL statement:** Opens the Specify SQL Statement screen in the wizard, enabling you to specify the connection string and an SQL statement.

Specify SQL Statement Screen

The Specify SQL Statement screen enables you to manually specify the database connection string and the SQL statement.



Enter the required information:

- **Connection String:** Enter the connection string, or click the **Create** button
- **Create:** Opens the ODBC Select Data Source dialog box. You can select a *.dsn file in the Select Data Source dialog box to have it insert the connection string in the box for you.
- **SQL:** Enter the SQL statement.

The Match Database Field screen enables you to identify the application control or text in your application that matches the displayed database field. You repeat this step for each field included in your query. This screen includes the following options:

- **Database field:** Displays a database field from your query. Use the pointing hand to identify the control or text that matches the displayed field name.
- **Logical name:** Displays the logical name of the control you select on your application.

(Displayed only when the **Select text from a Web page** check box is cleared.)

- **Text before:** Displays the text that appears immediately before the text to check.
(Displayed only when the Select text from a Web page check box is checked.)
- **Text after:** Displays the text that appears immediately after the text to check.
(Displayed only when the **Select text from a Web page** check box is selected.)

- **Select text from a Web page:** Enables you to indicate the text on your Web page containing the value to be verified.

The Matching Record Criteria screen enables you to specify the number of matching database records required for a successful checkpoint.



- **Exactly one matching record:** Sets the checkpoint to succeed if exactly one matching database record is found.
- **One or more matching records:** Sets the checkpoint to succeed if one or more matching database records are found.
- **No matching records:** Sets the checkpoint to succeed if no matching database records are found.

When you click **Finish** on the Runtime Record Checkpoint wizard, a **db_record_check** statement is inserted into your script.

Comparing Data in Different Formats

Suppose you want to compare the data in your application to data in the database, but the data is in different formats. You can follow the instructions below to create a runtime database record checkpoint without using the Runtime Record Checkpoint Wizard. Note that this feature is for advanced WinRunner users only.

For example, in the sample Flight Reservation application, there are three radio buttons in the Class box. When this box is enabled, one of the radio buttons is always selected. In the database of the

sample Flight Reservation application, there is one field with the values 1, 2, or 3 for the matching class.

To check that data in the application and the database have the same value, you must perform the following steps:

1. Record on your application up to the point where you want to verify the data on the screen. Stop your test. In your test, manually extract the values from your application.
2. Based on the values extracted from your application, calculate the expected values for the database. Note that in order to perform this step, you must know the mapping relationship between both sets of values. See the example below.
3. Add these calculated values to any edit field or editor (e.g. Notepad). You need to have one edit field for each calculated value. For example, you can use multiple Notepad windows, or another application that has multiple edit fields.
4. Use the GUI Map Editor to teach WinRunner:
 - the controls in your application that contain the values to check
 - the edit fields that will be used for the calculated values
5. Add TSL statements to your test script to perform the following operations:
 - extract the values from your application
 - calculate the expected database values based on the values extracted from your application
 - write these expected values to the edit fields
6. Use the Runtime Record Checkpoint wizard, described in “Using the Runtime Record Checkpoint Wizard,” to create a **db_record_check** statement.

When prompted, instead of pointing to your application control with the desired value, point to the edit field where you entered the desired calculated value.

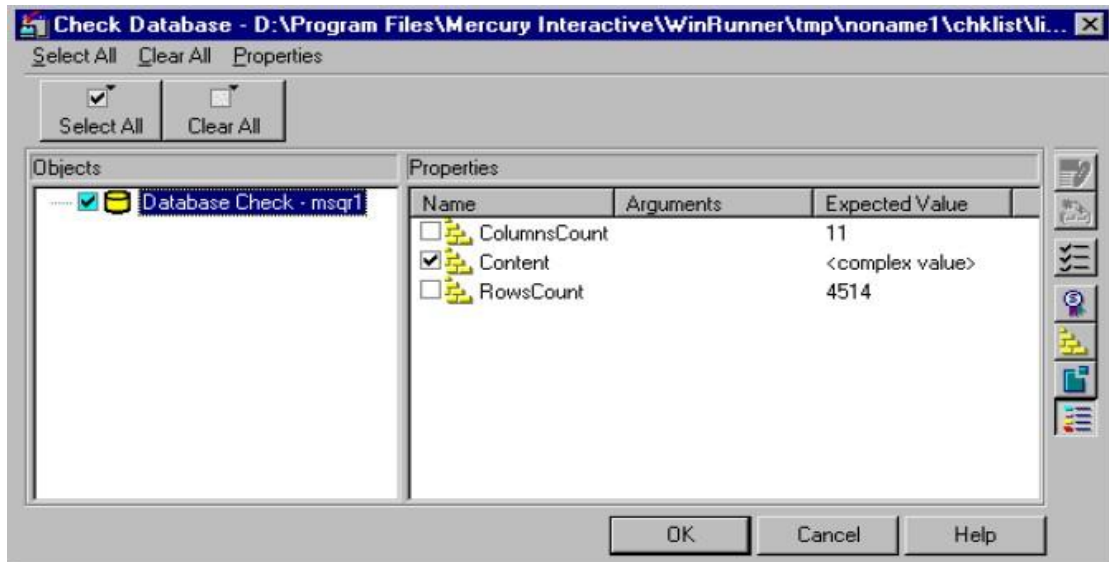
Example of Comparing Different Data Formats in a Runtime Database Record Checkpoint

The following excerpts from a script are used to check the Class field in the database against the radio buttons in the sample Flights application. The steps refer to the instructions.

step 1




```
# Extract values from GUI objects in application. button_get_state("First",vFirst);
button_get_state("Business",vBusiness); button_get_state("Economy",vEconomy);
```



The **Objects** pane contains “Database check” and the name of the *.sql query file or *.djs conversion file included in the database checkpoint. The **Properties** pane lists the different types of checks that can be performed on the result set. A check mark indicates that the item is selected and is included in the checkpoint.

7. Select the types of checks to perform on the database. You can perform the following checks:

ColumnsCount: Counts the number of columns in the result set.

Content: Checks the content of the result set, as described in “Creating a Default Check on a Database,”

RowCount: Counts the number of rows in the result set.

If you want to edit the expected value of a property, first select it. Next, either click the **Edit Expected Value** button, or double-click the value in the Expected Value column.

- For **ColumnsCount** or **RowCount** checks on a result set, the expected value is displayed in the **Expected Value** field corresponding to the property check. When you edit the expected value for these property checks, a spin box opens. Modify the number that appears in the spin box.

- For a **Content** check on a result set, <complex value> appears in the **Expected Value** field corresponding to the check, since the content of the result set is too complex to be displayed in this column. When you edit the expected value, the **Edit Check** dialog box opens. In the **Select Checks** tab, you can select which checks to perform on the result set, based on the data captured in the query. In the **Edit Expected Data** tab, you can modify the expected results of the data in the result set.
8. Click **OK** to close the Check Database dialog box.

WinRunner captures the current property values and stores them in the test's exp folder. WinRunner stores the database query in the test's chklist folder. A database checkpoint is inserted in the test script as a **db_check** statement.

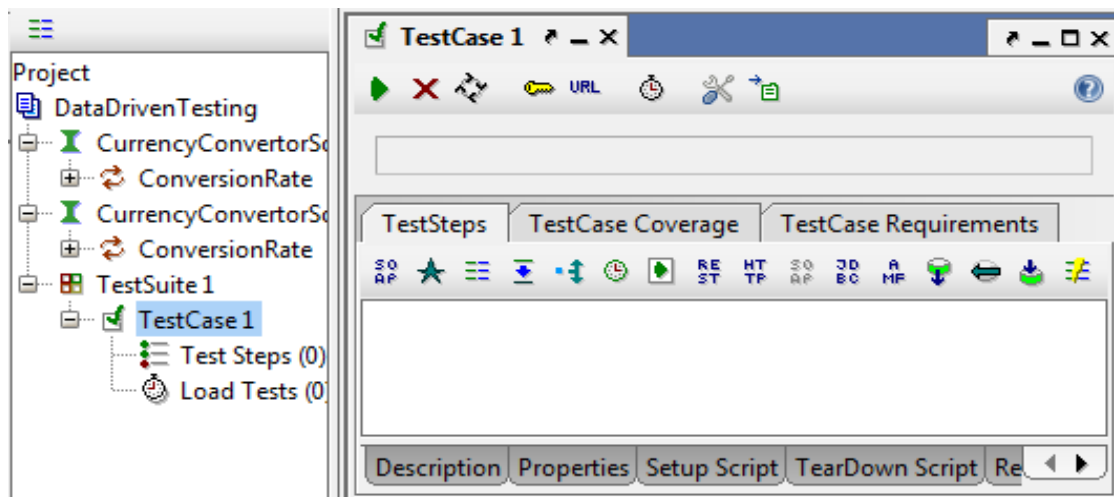
EXPERIMENT 9

B. Data driven test through flat files

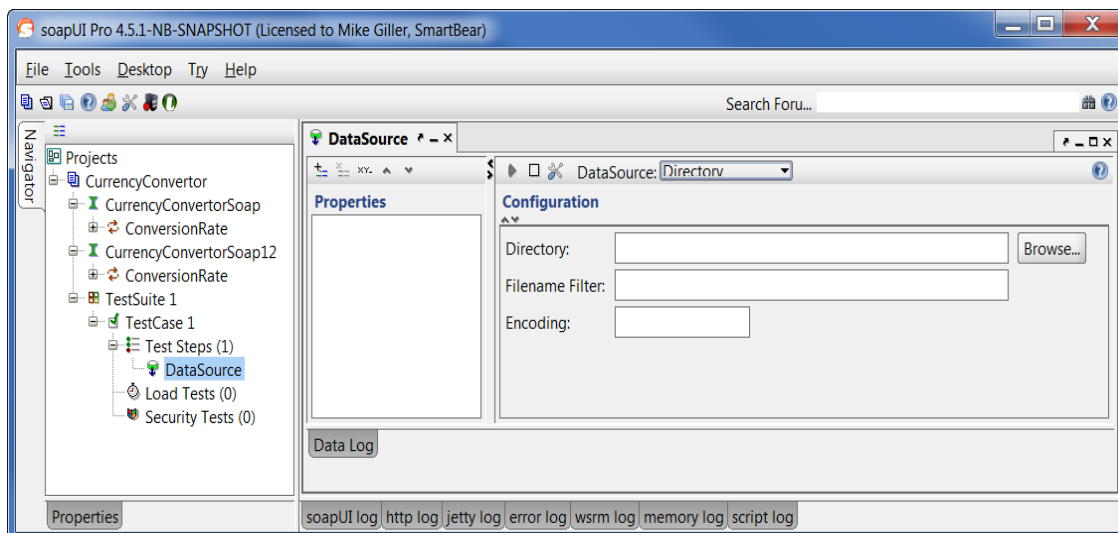
Steps:

1. Create DataSource

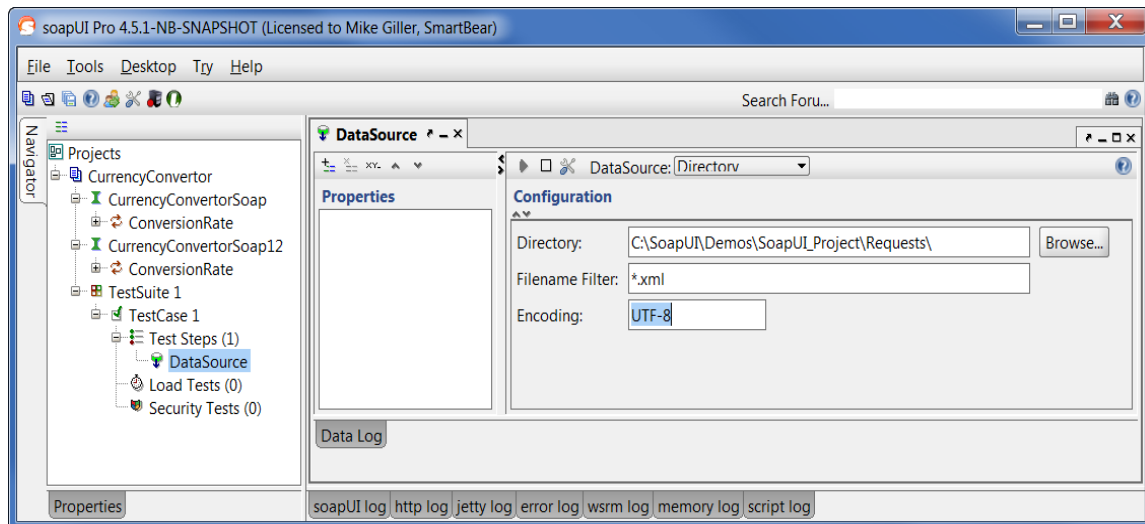
As in the Data Driven Testing guide, create a SoapUI Project from the publicly available CurrencyConverter WSDL (<http://www.webservices.com/CurrencyConverter.asmx?wsdl>), then add a TestSuite and a TestCase and open its editor:




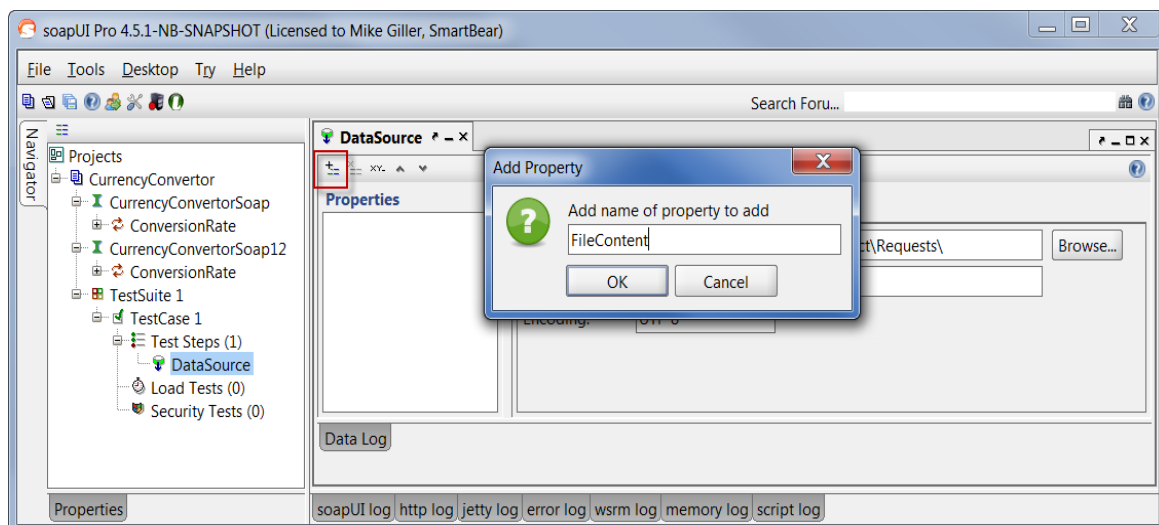
Now add a DataSource TestStep and select the DataSource type “Directory” from the dropdown in the toolbar. You should now have:



Now, select the directory where your input files are stored, add an applicable filter (e.g. “*.txt” or “*.xml” for text or XML files respectively), and potentially encoding.



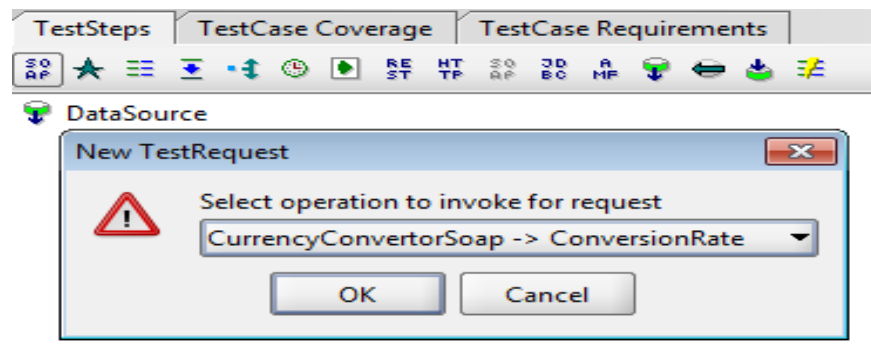
Now click on the  icon from the screen below and enter a property that will contain the content of each file



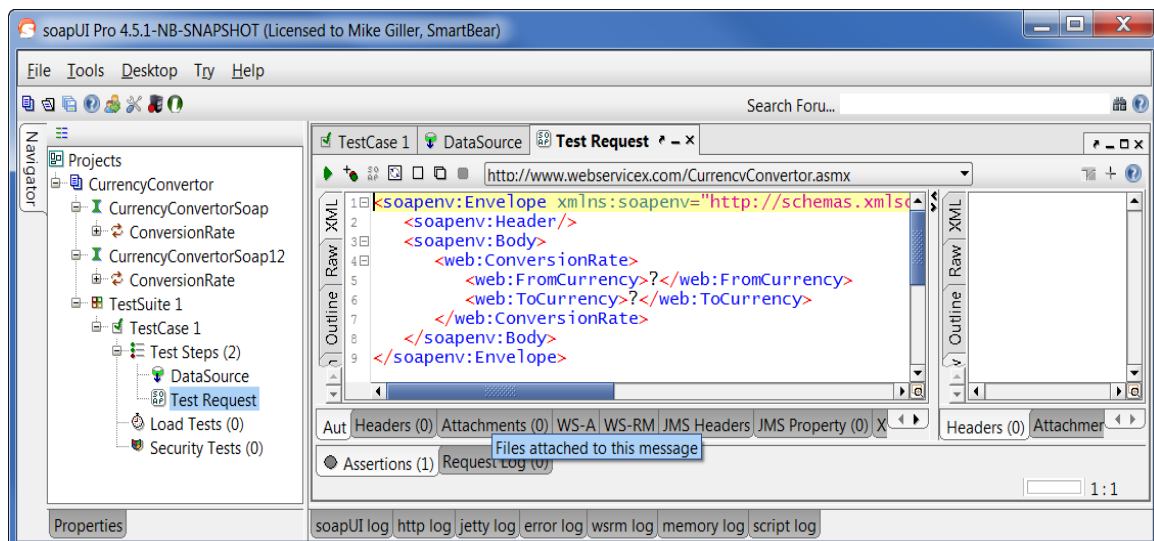
Quick tip: If your property is named “Filename” it will contain the name of the file instead of file’s contents.

2. Create Test Steps

Now you need to add a Test Request to your TestCase which you will use to test the Web Service. Press the SOAP Request button in the TestCase editor and select the ConversionRate operation in the CurrencyConverterSoap Interface.



Press OK in all dialogs. A SOAP Request Step will be added to the TestCase and the editor for the request is opened. Switch to the XML editor (if not already there):

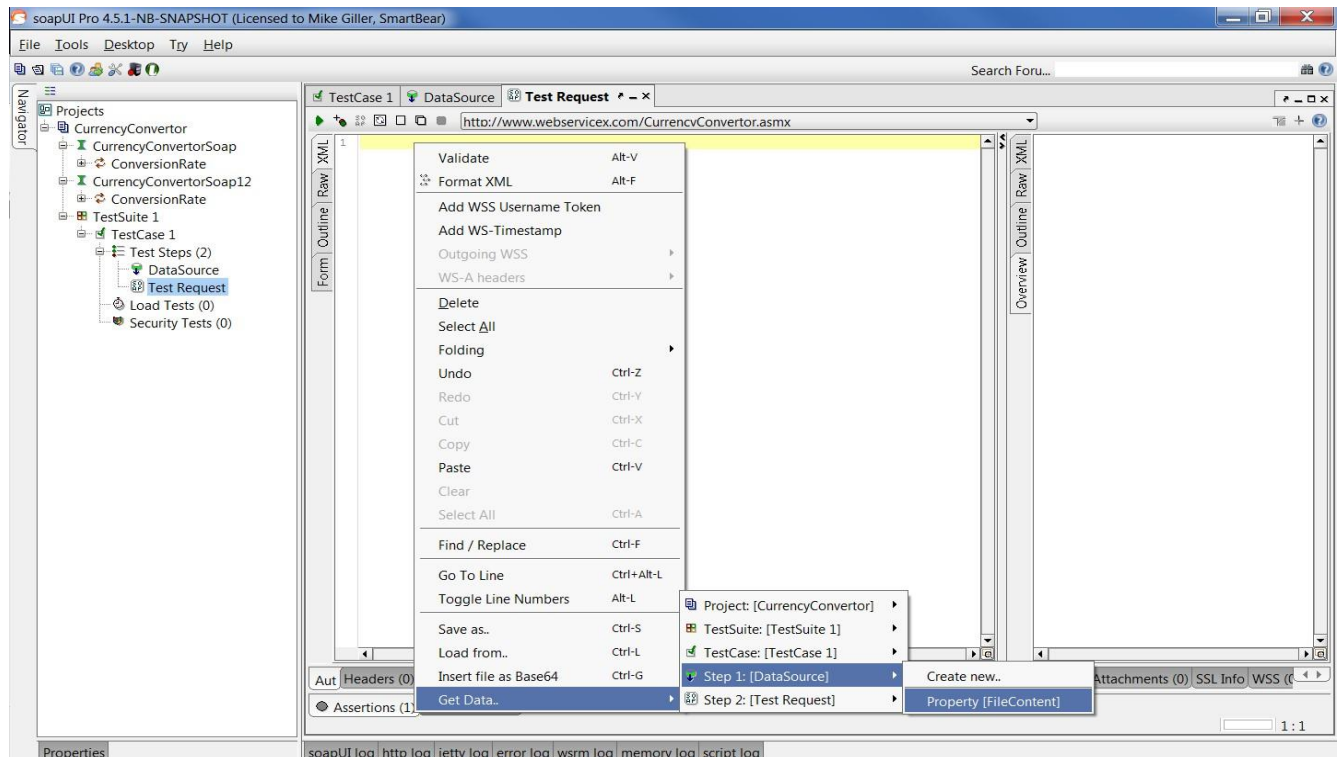


Now, I'm operating under the assumption that you have a fully built request in each of the files in your directory.

An example of an input file would be:

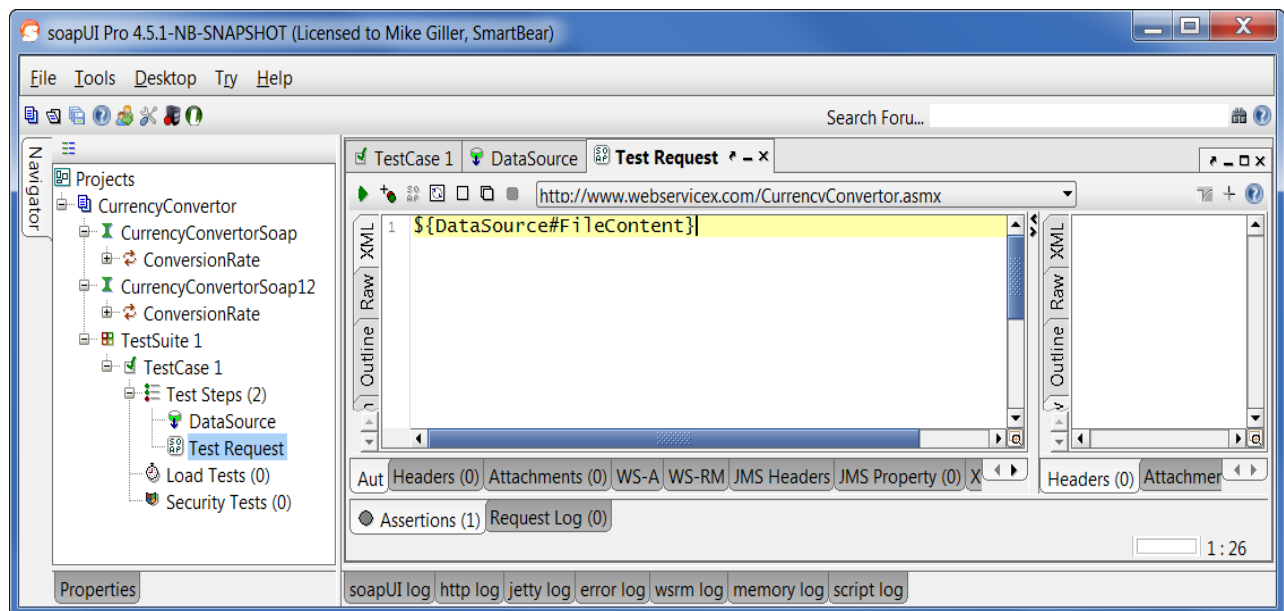
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://www.webserviceX.NET">
<soapenv:Header/>
<soapenv:Body>
<web:ConversionRate>
<web:FromCurrency>SEK</web:FromCurrency>
<web:ToCurrency>USD</web:ToCurrency>
</web:ConversionRate>
</soapenv:Body>
</soapenv:Envelope>
```

So, based on that, remove all the content in the XML tab, right-click and select the path to your DataSource property:



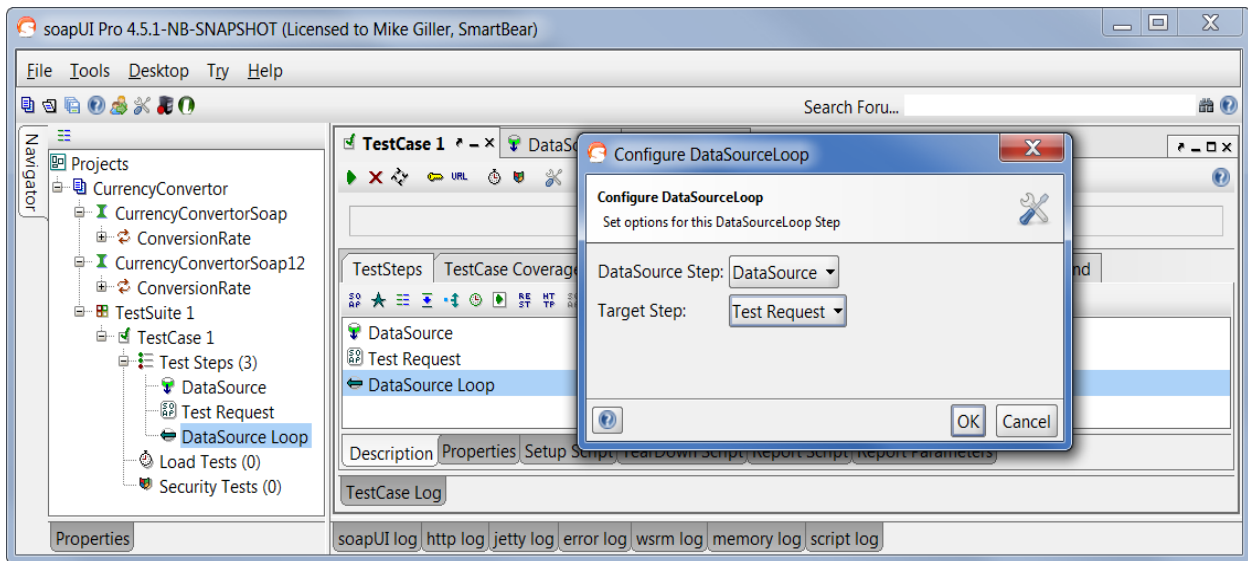
Note: If an XPATH window comes up, just click OK without selecting anything.

Now your request should look like this:




3. Add DataSource Loop

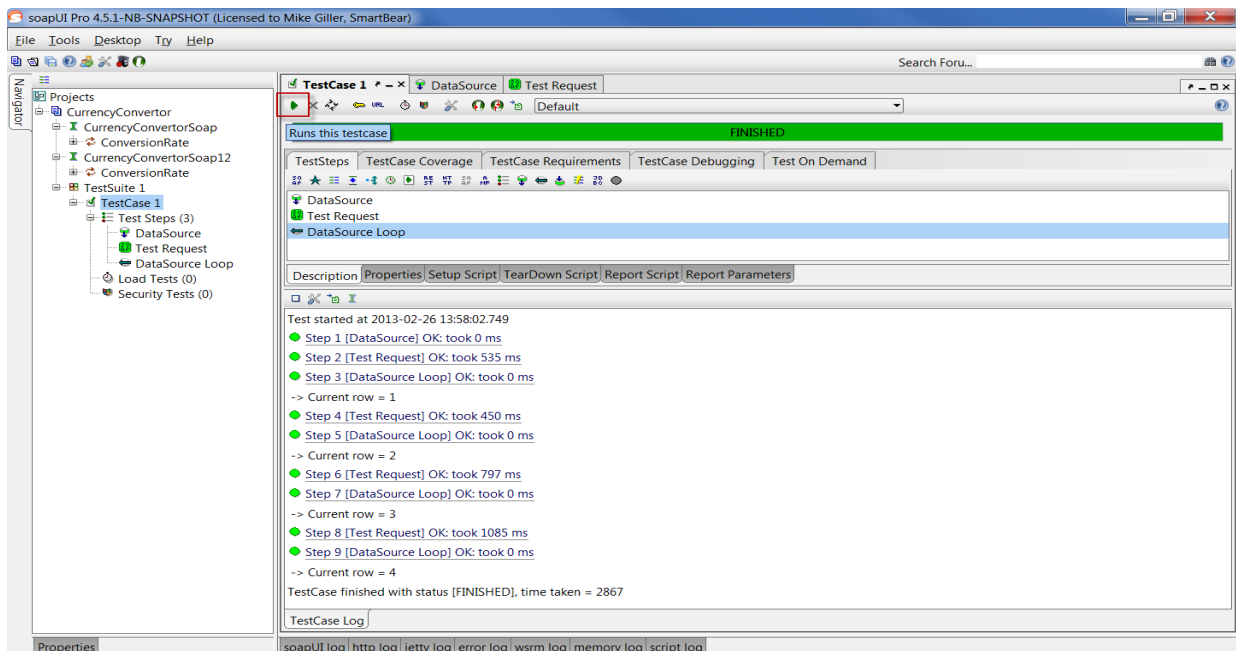
As a final step, we just need to iterate through all the files in our DataSource. So in your TestCase, add a DataSource loop step, and double click it to configure as in the picture below:



Click OK.

4. That's it

Now if you click on the  icon in the test case window, you can see the whole test run through each file:



C. Data driven test through excel test

Data-driven testing (DDT) is taking a test, parameterizing it and then running that test with varying data. This allows you to run the same test case with many varying inputs, therefore increasing coverage from a single test. In addition to increasing test coverage, data driven testing allows the ability to build both positive and negative test cases into a single test. Data-driven testing allows you to test the form with a different set of input values to be sure that the application works as expected.

It is convenient to keep data for automated tests in special storages that support sequential access to a set of data, for example, Excel sheets, database tables, arrays, and so on. Often data is stored either in a text file and are separated by commas or in Excel files and are presented as a table. If you need to add more data, you simply modify the file either in any text editor or in Microsoft Excel (in case of hard-coded values, you should modify both data and code).

Data-driven test includes the following operations performed in a loop:

- Retrieving input data from storage
- Entering data in an application form
- Verifying the results
- Continuing with the next set of input data

Pre-requisites:

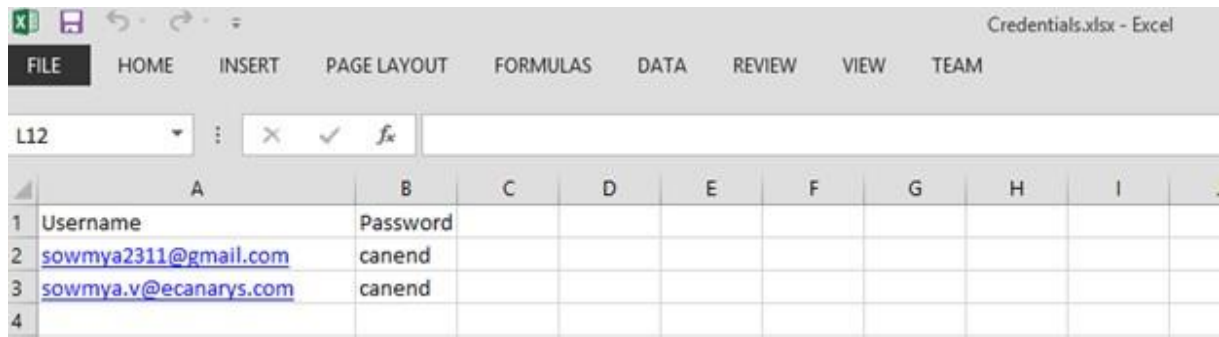
1. Java JDK 1.5 or above
2. Apache POI library v3.8 or above
3. Eclipse 3.2 above
4. Selenium server-standalone-2.47.x.jar
5. TestNG-6.9

Data Excel

Scenario -Open the application and login with different username and password. This data will be coming from excel sheet

Step 1: The first and the foremost step is to create the test data with which we would be executing the test scripts. Download JAR files of Apache POI and Add Jars to your project library. Let us create an excel file and save it as “Credentials.xlsx” and place it in the created package location.

We will use the data excel file and the file looks as below



	A	B	C	D	E	F	G	H	I
1	Username	Password							
2	sowmya2311@gmail.com	canend							
3	sowmya.v@ecanarys.com	canend							
4									

Step 2: Create a POM class file under com.coe.pom name it as “Loginpage.java”. Inside a login page we write code to identify the webelements of login page using @FindBy annotation. To initialize the web elements we use initelements of page factory class. We utilize the elements by writing a method to it.

Step 3: Create a ‘New Class’file, by right click on the package com.coe.script and select New > Class and name it as “SuperClass.java”, and create a new class file with a name “ValidLoginLogout.java”.

Step 4: Create some test data in excel that we will pass to script. For demo purpose I have taken username and password in excel.

Step 5: Copy and paste the below mentioned code under com.coe.pom package class.

EXPERIMENT 10

A. Batch testing without parameter passing

A batch test is a test script that calls other tests. You program a batch test by typing call statements directly into the test window and selecting the **Run in batch mode** option in the **Run** category of the General Options dialog box before you execute the test.

A batch test may include programming elements such as loops and decisionmaking statements. Loops enable a batch test to run called tests a specified number of times. Decision-making statements such as if/else and switch condition test execution on the results of a test called previously by the same batch script. See “Enhancing Your Test Scripts with Programming,” for more information.

For example, the following batch test executes three tests in succession, then loops back and calls the tests again. The loop specifies that the batch test should call the tests ten times.

```
for (i=0; i<10; i++)  
{  
  call "c:\\pbtests\\open" ();  
  call "c:\\pbtests\\setup" ();  
  call "c:\\pbtests\\save" ();  
}
```

To enable a batch test:

1. Choose **Tools > General Options**.

The General Options dialog box opens.

2. Click the **Run** category.
3. Select the **Run in batch mode** check box.

Running a Batch Test: You execute a batch test in the same way that you execute a regular test. Choose a mode (Verify, Update, or Debug) from the list on the toolbar and choose **Test > Run from Top**. See “Understanding Test Runs,” for more information.

When you run a batch test, WinRunner opens and executes each called test. All messages are suppressed so that the tests are run without interruption. If you run the batch test in **Verify** mode, the current test results are compared to the expected test results saved earlier. If you are running the batch test in order to update expected results, new expected results are created in the expected results folder



for each test. See “Storing Batch Test Results” below for more information. When the batch test run is completed, you can view the test results in the Test Results window.

Note that if your tests contain TSL **texit** statements, WinRunner interprets these statements differently for a batch test run than for a regular test run. During a regular test run, **texit** terminates test execution. During a batch test run, **texit** halts execution of the current test only and control is returned to the batch test.

EXPERIMENT 11

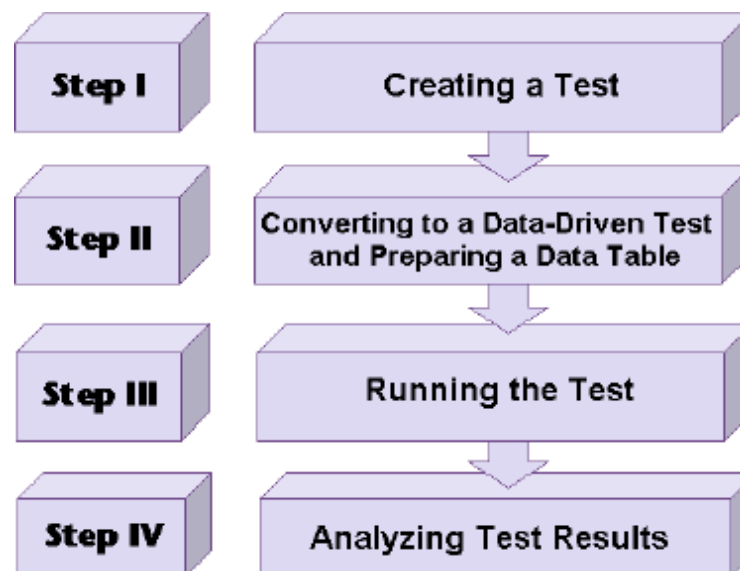
Data driven batch

When you test your application, you may want to check how it performs the same operations with multiple sets of data. For example, suppose you want to check how your application responds to ten separate sets of data. You could record ten separate tests, each with its own set of data.

Alternatively, you could create a data-driven test with a loop that runs ten times. In each of the ten iterations, the test is driven by a different set of data. In order for WinRunner to use data to drive the test, you must substitute fixed values in the test with parameters. The parameters in the test are linked with data stored in a data table. You can create data-driven tests using the DataDriver wizard or by manually adding data-driven statements to your test scripts.

For non-data-driven tests, the testing process is performed in three steps: creating a test; running the test; analyzing test results. When you create a data-driven test, you perform an extra two-part step between creating the test and running it: converting the test to a data-driven test and creating a corresponding data table.

The following diagram outlines the stages of the data-driven testing process in WinRunner:



EXPERIMENT 12

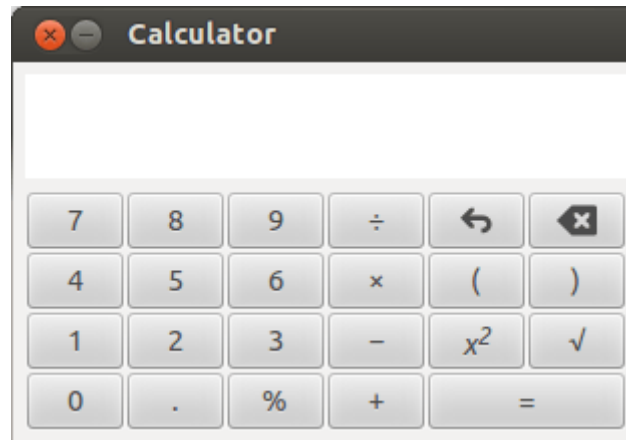
Silent mode test execution without any interruption

1 **Silent Mode** to continue **test execution without any interruption**, we can use this run setting.

1 Navigation Click Tools menu → Choose General options→ Select Run Tab → Select Run in batch mode (Figure III . 18 . 1) – Click ... NOTE In **silent mode** , **WinRunner** is **not** able to **execute** tester interactive statements .

EXPERIMENT 13

Test case for calculator in windows application



Basic Operational Tests

Write the test cases based on the following functions and scenarios.

- Check the calculator if it starts by on button. If it is software based calculator, then check if it starts via specific means like from searching for calculator in search bar and then executing application. Or by accessing menu item in the Windows.
- Check if the calculator window maximizes to certain window size.
- Check the if the calculator closes when the close button is pressed or if the exit menu is clicked from file > exit option.
- Check if the help document is accessed from Help > Documentation.
- Check if the calculator allows copy and paste functionality.
- Check if the calculator has any specific preferences.
- Check if all the numbers are working (0 to 9)
- Check if the arithmetic keys (+, -, *, %, /) are working.
- Check if the clear key is working.
- Check if the brackets keys are working.
- Check if the sum or equal key is working.
- Check if the square and square root key is working.

Functionality Test Cases

- Check the addition of two integer numbers.
- Check the addition of two negative numbers.
- Check the addition of one positive and one negative number.
- Check the subtraction of two integer numbers.
- Check the subtraction of two negative numbers.
- Check the subtraction of one negative and one positive number.
- Check the multiplication of two integer numbers.
- Check the multiplication of two negative numbers.
- Check the multiplication of one negative and one positive number.
- Check the division of two integer numbers.
- Check the division of two negative numbers.
- Check the division of one positive number and one integer number.
- Check the division of a number by zero.
- Check the division of a number by negative number.
- Check the division of zero by any number.
- Check if the functionality using BODMAS/BIDMAS works as expected.

Test Step	Calc 1
Requirement	1
Input	Start the computer, click on start button, select 'programs' go to 'Accessories' and then click on 'Calculator'
Expected Output	System should display 'Calculator' window
Test Step	Calc 2
Requirement	1.1
Input	Click on start button and then click on 'Run' enter 'calc.exe' and click on OK button
Expected Output	System should display 'Calculator' window
Test Step	Calc 3
Requirement	1.2
Input	Click on start button and then click on 'Run' enter 'xyz.exe' and click on OK button
Expected Output	System should display an error message 'Can't find the xyz .exe, make sure the path and file name are correct and that all required libraries are available
Test Step	Calc 4
Requirement	1.3
Input	On 'Calculator' check for all buttons either its working properly or not [Ex: buttons: 1 to 9, +, -, *, /, etc]
Expected Output	All buttons should work fine according to requirements.
Test Step	Calc 5
Requirement	1.4
Input	Check for following conditions Additions, subtractions, Multiplications and division.
Expected Output	Calculator should work fine according to stated requirements
Test Step	Calc 6
Requirement	1.5
Input	Check for following condition 1. Zero divided by some number [Ex: 1,20,30,4,5,67...etc] and then press equal '=' button
Expected Output	Calculator should display value zero

EXPERIMENT 14

TEST CASES FOR MOBILE APPLICATION TESTING

We can test the efficiency of the Mobile Applications in different ways. Following are some of the types to be considered and develop the test cases

1. USABILITY TEESTING: Check whether...

- Responsiveness of the application name and logo when the Application Manager is clicked
- Receiving visual review for the user activities in the app within just three seconds maximum.
- The functionality of exit options at any point when the app is running
- Keep away the unmapped keys
- Enabling highly responsive Mobile menu for Mobiles and Tablets.
- Easy navigation across various screens.

2. PERFORMANCE TESTING: Check whether...

- Time is taken for launching the app.
- App performance during peak load scenarios
- Splash performance test and making sure it stays on the screen for no more than three to four seconds.
- App performance when charging and during low battery conditions.
- Leverage Live Monitoring solutions for keeping the computing power of the application on the check.
- Successful installing and uninstalling of the app within the stipulated timeframe.
- Graceful exit and display of error messages during low memory conditions
- Performance of the app during network issues and prompts for error alerts.
- App performance when the network resumes into action.

3. ACCESSIBILITY TESTING: Checking whether...

- Screen reader testing
- Zooming the application
- Verification of the colour ratios
- Readability testing of the application
- Ensure navigation is structured, consistent, descriptive, and logical

4. SECURITY TESTING: Check whether...

- Security of the users' payment data
- Security of Network protocols for running apps
- A breach in app's security as well as error reporting
- Authentication of the app permissions and certificates

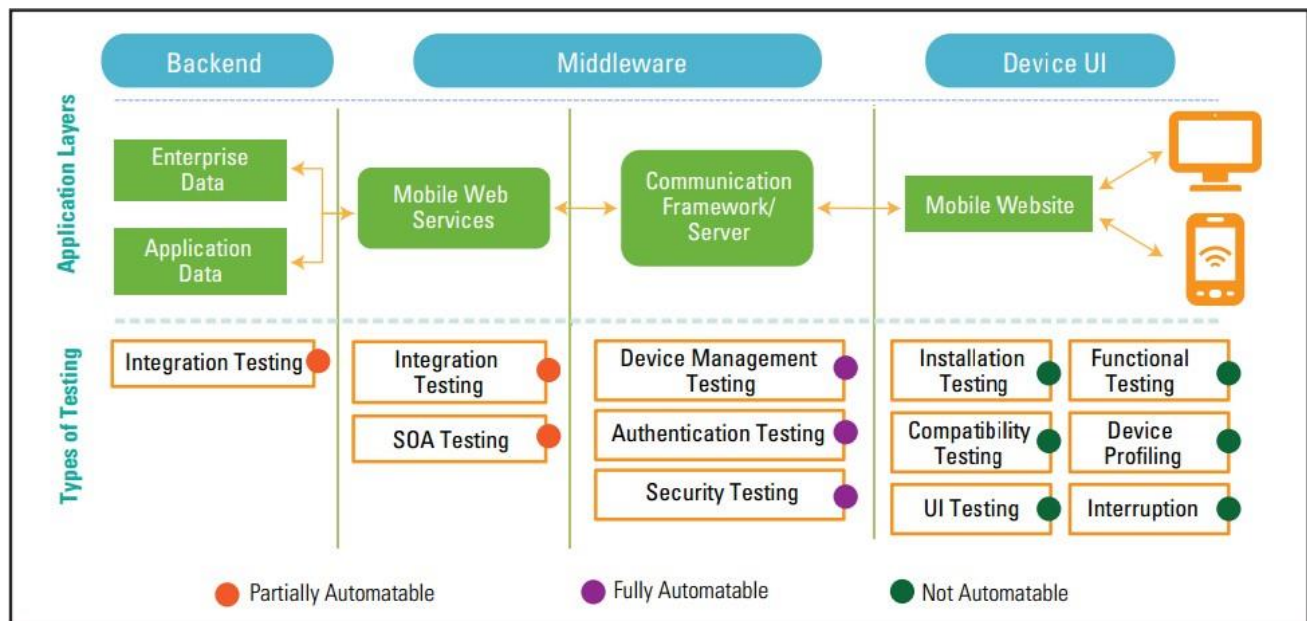
Main Test Cases to be considered while performing the Testing on Mobile Applications:

- Ensure the app has been launched by downloading and installing it for use.
- Verify that the mobile app display is adaptable to the device screen and also ensures all menus on the app are functioning.
- Verify that the text on the **mobile app** is readable and clear.
- Check that the app display is adaptable and amenable to various display modes (landscape /portrait).
- Verify that the app does not stop the functioning of other apps in the mobile device.
- Verify that in the play screen, the back key allows the app to go back to the start-up screen.
- check that the app still operates as intended, if the device resumes from inactive mode or from the lock screen.
- Check whether the app reminds the user to save setting changes or changing of information before moving to other activities on the app.
- Verify that the on-screen keyboard appears immediately the user attempt to enter a text.
- Check if the app behaves as designed if the mobile device is shaken
- Verify that the app still functions as designed when “battery low” notifications appear on the screen.
- Check that the app goes into the background when on call
- Check that the app still operates as designed when a message or notification pop-up from another app such as Face book messaged, Instagram, etc.
- If the app comes with users’ settings features, check if the app changes when some form of changes is affected by the user.
- Check the **Performance** of the app on the different internet networks such as 1G, 2G, 3G or 4 G network.
- Check that the app operates as intended when the device is connected to the internet through WiFi.
- Check that the app still operates normally when there is an incoming call or SMS.
- Check that the app is adaptable to different mobile platforms or OS, like Android, iOS, Microsoft, etc.
- Check that the font size and style of the app are compatible and readable to the users
- Verify that that the loading time for the app is not too long.
- Check that the app is still working as intended after the successful update of the app.
- Check how the app functions under different battery levels and temperatures.
- Verify that the app is not draining too much battery.
- Check that the app support image capturing.
- Check that the app does not log-out the user before the end of a session.

Example of Test cases:

Feature Name	Story ID	Test Case ID	Summary	Precondition	Execution Steps	Expected Result
Install	Mob-1214	Mob-1214_1	Verify that application should be Installed successfully.	1. Select the application from google play.	1. Click on install button. 2. Navigate to the menu and click on the newly installed app.	The application should be Installed successfully.
Uninstall	Mob-1214	Mob-1214_2	Verify that application should be Uninstalled successfully.	1. Execute test case ID: Mob-1214_1	1. Click on settings. 2. Select the newly added application on Mob-1214_1. 3. Click on Uninstall button. 4. Verify.	The application should be Uninstalled successfully.
Interruption by Calls	Mob-1214	Mob-1214_3	Verify that user should able to accept Phone calls when application is running and should continue from the same point.	1. Execute test case ID: Mob-1214_1	1. Open the application. 2. Navigate here an there for a moment. 3. Make a call from another device to the device where you have opened the application. 4. Pick up the call. 5. Now disconnect it and verify.	User should able to accept Phone calls when application is running and should continue from the same point.
Interruption by Messages	Mob-1214	Mob-1214_4	Verify that user should able to accept messages when application is running and should continue from the same point after reading the message.	1. Execute test case ID: Mob-1214_1	1. Open the application. 2. Navigate here an there for a moment. 3. Send a message from another device to the device where you have opened the application. 4. Read the message. 5. Close the message app and verify.	User should able to accept messages when application is running and should continue from the same point after reading the message.
Memory	Mob-1214	Mob-1214_5	Verify that user should able to see proper error message when device memory is low.	1. The device memory space should not more than 20 mb. 2. Execute test case ID: Mob-1214_1 Note: Application memory	1. Go to the app from google play store. 2. Click on the install button. 3. Wait till the application get installed and verify.	Application should display with proper error message when device memory is low.

				requirement is 25 MB.		
Exit application	Mob-1214	Mob-1214_6	Verify that user should able to exit from application if we click on end key.	1. Execute test case ID: Mob-1214_1	1. Click on app and open it. 2. Now press the end key and verify.	User should able to exit from application if we click on end key.
Battery	Mob-1214	Mob-1214_7	Verify that user should able to see the alert when battery is low.	1. Execute test case ID: Mob-1214_1	1. Click on app and open it. 2. Use the application till you get the low battery indication.	When battery is low the alert should display.
Battery Consumption	Mob-1214	Mob-1214_8	Verify that application should not consume more battery	1. Execute test case ID: Mob-1214_1 2. Full charge your device.	1. Click on app and open it. 2. Use the application and verify the status of the battery in 15 mins interval of time.	Application should not consume more battery
Charge	Mob-1214	Mob-1214_8	Verify that application should run when inserting the charger. It will not affect the application	1. Execute test case ID: Mob-1214_1	1. Click on app and open it. 2. Insert the charging pin in between running of the application and verify.	Application should run when inserting the charger. It will not affect the application



EXPERIMENT 15

TEST CASES FOR CLOUD ENVIRONMENT TESTING

Sl. NO	Test Scenarios	Test cases
1	Performance Testing	Failure due to one user action on the cloud should not affect other users performance
		Manual or automatic scaling should not cause any disruption
		On all types of devices, the performance of the application should remain the same
		Overbooking at supplier end should not hamper the application performance
2	Security Testing	An only authorized customer should get access to data
		Data must be encrypted well
		Data must be deleted completely if it is not in use by a client
		Data should be accessible with insufficient encryption
		Administration on suppliers end should not access the customers' data
		Check for various security settings like firewall, VPN, Anti-virus etc.
3	Functional testing	Valid input should give the expected results
		Service should integrate properly with other applications
		A system should display customer account type when successfully login to the cloud
		When a customer chose to switch to other services the running service should close automatically
4	Interoperability & Compatibility Testing	Validate the compatibility requirements of the application under test system
		Check browser compatibility in a cloud environment
		Identify the Defect that might arise while connecting to a cloud
		Any incomplete data on the cloud should not be transferred
		Verify that application works across a different platform of cloud
		Test application on the in-house environment and then deploy it on a cloud environment
5	Network Testing	Test protocol responsible for cloud connectivity
		Check for data integrity while transferring data
		Check for proper network connectivity
		Check if packets are being dropped by a firewall on either side
6	Load and Stress Testing	Check for services when multiple users access the cloud services
		Identify the Defect responsible for hardware or environment failure
		Check whether system fails under increasing specific load
		Check how a system changes over time under a certain load

dotcom-monitor
Stress Testing
STRESS TESTS
TOOLS
Chat Now
Solutions
Support
Account

Load Test Scenario: dotcom-monitor.com

Load Device: dotcom-monitor.com
ADJUST USER BEHAVIOR
EDIT DEVICE

Task Type	Task Name	URL	Status
> BrowserView	https://www.dotcom-monitor.com	https://www.dotcom-monitor.com	OK

Validation Result: ✓ OK [View Details](#)
VALIDATE

Load Step Curve
Generates loads with a pre-determined number of concurrent users for specified time durations.

Goal-Based Curve
Automatically adjusts concurrent users to reach a required rate of transactions per time interval.

Dynamic Adjustable Curve
Manually adjust concurrent users in real-time, while the test is running.

Stress Test Scenario Steps

#	Start At	Action	Duration
1	00:00	Start With <input type="text" value="10"/> users	
2	00:00	Hold For <input type="text" value="5"/> min	
3	05:00	Raise By <input type="text" value="10"/> users/min <input type="text" value="5"/> min	
4	10:00	Hold For <input type="text" value="5"/> min	

Load Curve

Test Duration: 42 min
Max Users: 210
Estimated Sessions: —
Wallet Balance: \$500.00
Cost for this Test: \$184.95
Validation Result: OK
RUN TEST →

EXPERIMENT-16

SAMPLE TEST CASES FOR A PEN

General Test Cases/Scenarios for all Types of Pen:

1. **The grip of the pen:** Verify if you are able to hold the pen comfortably.
2. **Writing:** Verify if you are able to write smoothly.
3. Verify that the pen is not making any sound while writing.
4. Verify the ink flow. It should not overflow nor get a break either.
5. Verify the quality of the material used for the pen.
6. Verify if the company or pen name is visible clearly.
7. Verify if the pen color or text written on the pen is not getting removed easily.
8. Verify, whether the width of the line drawn by the pen is as per the expectations or not.
9. Verify the ink color, it should be consistent from the start till the end.
10. Verify if a pen can write on a variety of papers like smooth, rough, thick, thin, glossy etc.
11. Verify for the waterproof ink. [Not for gel and ink pens].
12. Verify if the ink will not get dried easily by keeping the pen open for some time. [Not for ink pen]
13. Verify if any other refill fits in the pen or not.
14. Verify that the pen doesn't have sharp edges or corners.
15. Verify if the ink and external assembly of the pen is made of non-toxic material.

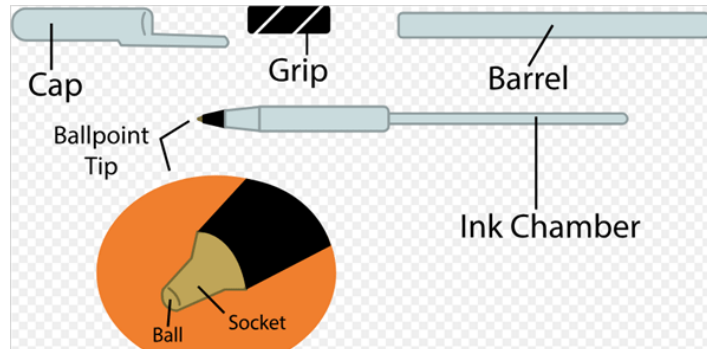
Negative Test Cases/Scenarios

1. Put the pen in water and then try to write. Verify if you are able to write with this pen. The pen can get wet because of the water spill on the table or during the rainy season. It can be due to any reason.
2. Drop the pen from some height (Table height) in the upside-down position. Verify if you are able to write with this pen. By mistake, the pen can any time fall on the ground. So testing this possibility to know its impact, will help us in knowing the quality of the pen.

Performance Test Cases/Scenarios

1. Verify how fast you can write with this pen.
2. Verify if the pen will perform the same even though you use continuously for hours.
3. Verify 'How much can be written in one refill/ink sac?'
4. Verify if the tip or nib of the pen is not destroyed after continuous writing for hours.

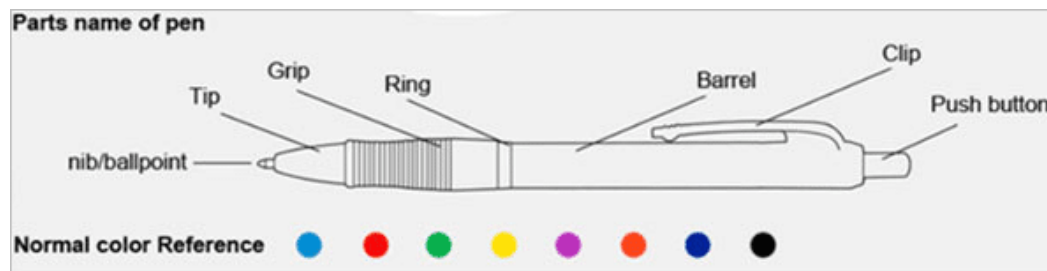
#1) Ball Pens with Cap:



Test Cases:

1. **Pen Cap:** Verify if the pen cap is tight enough so that it will not get removed easily.
2. Verify while holding in a pocket, if the pen cap is not getting removed.

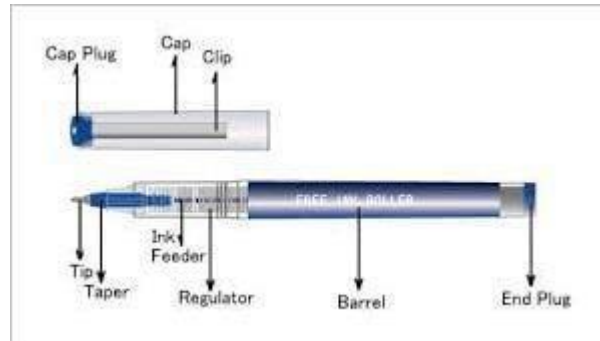
#2) Tick Tock ball pen:



Test Cases:

1. **Pen button:** Verify when the pen button is pressed, if the refill comes out and when pressed again it goes in.
2. Verify the on and off modes of the pen.
3. **Pen button:** Verify if the pen button will not get stuck if pressed continuously for 5 to 6 times.
4. Verify the pen clip, it should be tight enough to hold in a pocket.
5. Verify the tip of the pen, if you write by putting some pressure, then it should not get broken.
6. Verify if the tip is easy to open and close.

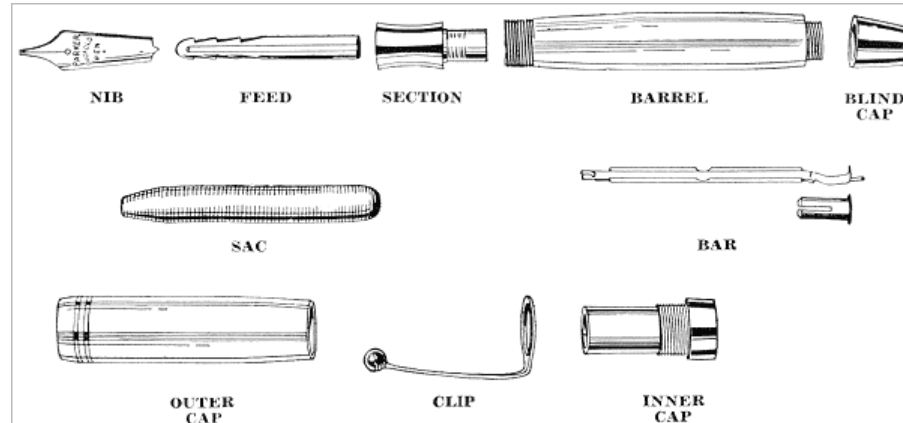
#3) Gel Pen:



Test Cases:

1. Verify if the ink should not get overflowed.
2. Verify that the ink should be dark enough. But at the same time, it should not be too dark that it will make an impression on the other side of the paper.
3. Ink should get dried quickly. It should not get spread easily with hand.
4. Verify if the Tip and End plug are getting opened and closed easily and correctly.

#4) Ink Pen:



Test Cases:

1. Verify the ink flow through the nib.
2. Verify that there is no ink leakage through the section.
3. Verify while refilling ink, the sac is getting full.
4. Verify for ink leakage by holding the pen in horizontal, vertical, and upside-down positions.

#5) Multi Refill pen:



Test Cases:

1. Verify all the buttons.
2. Verify if the button color (if the pen has different color buttons) is matching with refill color. [We are testing for the ease of use]
3. Verify if you can change the refill easily. It should not be a complicated process.
4. Verify the grip of this pen. Verify that the pen is not too bulky to hold.

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

VIVA-VOCE QUESTION & ANSWERS

1. What is Software Testing?

According to ANSI/IEEE 1059 standard – A process of analysing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item. Click [here](#) for more details.

2. What are the best practices for writing test cases?

- Write test cases with end-users' perspective
- Write test steps in a simple way that anyone can follow them easily
- Make the test cases reusable
- Set the priority
- Provide a test case description, test data, expected result, precondition, post condition.
- Write invalid test cases along with valid test cases
- Follow proper naming conventions
- Review the test cases regularly and update them if necessary.

3. What is configuration management?

Configuration management (CM) is a process of systems engineering to maintain system resources, computer systems, servers, software, and product's performance in a consistent state. It helps to record all the changes made in the system and ensures that the system performs as expected even though changes are made over time.

4. Name some popular configuration management tools?

Some of the popular configuration management tools are Ansible, Chef, Puppet, Terraform, Saltstack, etc.

5. What if the software is so buggy it can't really be tested at all?

If the software is so buggy, the first thing we need to do is to report the bugs and categories them based on Severity. If the bugs are critical bugs, then it severely affects schedules and indicates deeper problems in the software development process. So you need to let the manager know about the bugs with proper documentation as evidence.

6. What are Quality Assurance and Quality Control?

Quality Assurance: Quality Assurance involves in process-oriented activities. It ensures the prevention of defects in the process used to make Software Applications. So the defects don't arise when the Software Application is being developed.

Quality Control: Quality Control involves in product-oriented activities. It executes the program or code to identify the defects in the Software Application.

7. What is Verification and Validation in software testing?

Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not. Activities involved here are Inspections, Reviews, Walk-throughs. Click [here](#) for more details.

Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not. Activities involved in this is Testing the software application. Click [here](#) for more details.



8. What is the workbench concept in Software Testing?

Workbench is a practice of documenting how a specific activity must be performed. It is often referred to as phases, steps, and tasks.

In every workbench there will be five tasks such as Input, Execute, Check, Output, and rework.

9. What is Static Testing?

Static Testing involves reviewing the documents to identify the defects in the early stages of SDLC. In static testing, we do code reviews, walkthroughs, peer reviews, and static analysis of a source code by using tools like StyleCop, ESLint, etc.,

10. What is Dynamic Testing?

Dynamic testing involves the execution of code. It validates the output with the expected outcome.

11. What is White Box Testing?

White Box Testing is also called as Glass Box, Clear Box, and Structural Testing. It is based on applications internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing usually was done at the unit level. Click here for more details.

Various white-box testing techniques are:

- Statement Coverage
- Decision Coverage
- Condition Coverage
- Multiple Condition Coverage

12. What is Black Box Testing?

Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure. This can be applied to every level of software testing such as Unit, Integration, System and Acceptance Testing. Click here for more details.

13. What is Grey Box Testing?

Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.

14. What is Positive and Negative Testing?

Positive Testing: It is to determine what system supposed to do. It helps to check whether the application is justifying the requirements or not.

Negative Testing: It is to determine what system not supposed to do. It helps to find the defects from the software.

15. What is Test Strategy?

Test Strategy is a high-level document (static document) and usually developed by the project manager. It is a document that captures the approach on how we go about testing the product and achieve the goals. It is normally derived from the Business Requirement Specification (BRS). Documents like Test Plan are prepared by keeping this document as a base. Click here for more details.

16. What is Test Plan and contents available in a Test Plan?



Test plan document is a document which contains the plan for all the testing activities to be done to deliver a quality product. Test Plan document is derived from the Product Description, SRS, or Use Case documents for all future activities of the project. It is usually prepared by the Test Lead or Test Manager.

- Test plan identifier
- References
- Introduction
- Test items (functions)
- Software risk issues
- Features to be tested
- Features not to be tested
- Approach
- Items pass/fail criteria
- Suspension criteria and resolution requirements
- Test deliverables
- Remaining test tasks
- Environmental needs
- Staff and training needs
- Responsibility
- Schedule
- Plan risks and contingencies
- Approvals
- Glossaries
- [Click here for more details.](#)

17. What is Test Suite?

Test Suite is a collection of test cases. The test cases which are intended to test an application.

18. What is Test Scenario?

Test Scenario gives the idea of what we have to test. Test Scenario is like a high-level test case.

19. What is Test Case?

Test cases are the set of positive and negative executable steps of a test scenario which has a set of pre-conditions, test data, expected result, post-conditions and actual results. [Click here for more details.](#)

20. What is Test Bed?

An environment configured for testing. Test bed consists of hardware, software, network configuration, an application under test, other related software.

21. What is Test Environment?

Test Environment is the combination of hardware and software on which Test Team performs testing.

Example:

Application Type	: Web Application
OS	: Windows
Web Server	: IIS
Web Page Design	: Dot Net
Client Side Validation	: JavaScript
Server Side Scripting	: ASP Dot Net



Database : MS SQL Server
Browser : IE/FireFox/Chrome

22. What is Test Data?

Test data is the data that is used by the testers to run the test cases. Whilst running the test cases, testers need to enter some input data. To do so, testers prepare test data. It can be prepared manually and also by using tools. For example, To test a basic login functionality having a user id, password fields. We need to enter some data in the user id and password fields. So we need to collect some test data.

23. What is Test Harness?

A test harness is the collection of software and test data configured to test a program unit by running it under varying conditions which involves monitoring the output with the expected output. It contains the Test Execution Engine & Test Script Repository

24. What is Test Closure?

Test Closure is the note prepared before test team formally completes the testing process. This note contains the total no. of test cases, total no. of test cases executed, total no. of defects found, total no. of defects fixed, total no. of bugs not fixed, total no of bugs rejected etc.,

25. What are the tasks of Test Closure activities in Software Testing?

Test Closure activities fall into four major groups.

Test Completion Check: To ensure all tests should be either run or deliberately skipped and all known defects should be either fixed, deferred for a future release or accepted as a permanent restriction.

Test Artifacts handover: Tests and test environments should be handed over to those responsible for maintenance testing. Known defects accepted or deferred should be documented and communicated to those who will use and support the use of the system.

26. What is test coverage?

Test coverage helps in measuring the amount of testing performed by a set of tests.

Test coverage can be done on both functional and non-functional activities. It assists testers to create tests that cover areas which are missing.

27. What is Code coverage?

Code coverage is different from Test coverage. Code coverage is about unit testing practices that must target all areas of the code at least once. It is usually done by developers or unit testers.

28. List out Test Deliverables?

1. Test Strategy
2. Test Plan
3. Effort Estimation Report
4. Test Scenarios
5. Test Cases/Scripts
6. Test Data
7. Requirement Traceability Matrix (RTM)
8. Defect Report/Bug Report
9. Test Execution Report
10. Graphs and Metrics
11. Test summary report
12. Test incident report



13. Test closure report
14. Release Note
15. Installation/configuration guide
16. User guide
17. Test status report
18. Weekly status report (Project manager to client)
19. [Click here for more details.](#)

29. What are the most common components of a defect report?

The most common components of a defect report format include the following

- Project Name
- Module Name
- Defect ID
- Defect detected on
- Defect detected by
- Priority
- Severity
- Defect resolved on
- Defect resolved by

30. What are the levels of testing?

In software testing, there are four testing levels.

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

31. What is Unit Testing?

Unit Testing is also called Module Testing or Component Testing. It is done to check whether the individual unit or module of the source code is working properly. It is done by the developers in the developer's environment. Learn more about Unit Testing in detail.

32. What is Integration Testing?

Integration Testing is the process of testing the interface between the two software units. Integration testing is done in three ways. Big Bang Approach, Top-Down Approach, Bottom-Up Approach. Learn more about Integration Testing in detail.

33. What is System Testing?

Testing the fully integrated application to evaluate the system's compliance with its specified requirements is called System Testing AKA End to End testing. Verifying the completed system to ensure that the application works as intended or not.

34. What is Big Bang Approach?

Combining all the modules once and verifying the functionality after completion of individual module testing. Top-down and bottom-up are carried out by using dummy modules known as Stubs and Drivers. These Stubs and Drivers are used to stand in for missing components to simulate data communication between modules.

35. What is Top-Down Approach?

Testing takes place from top to bottom. High-level modules are tested first and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended. Stubs are used as a temporary module if a module is not ready for integration testing.

36. What is Bottom-Up Approach?

It is a reciprocal of the Top-Down Approach. Testing takes place from bottom to up. Lowest level modules are tested first and then high-level modules and finally integrating the high-level modules to a low level to ensure the system is working as intended. Drivers are used as a temporary module for integration testing.

37. What is End-To-End Testing?

In simple words, end-to-end testing is the process of testing software from start to end. Check this End-To-End Testing guide for more information. Also, refer System Testing tutorial.

38. What is Functional Testing?

In simple words, what the system actually does is functional testing. To verify that each function of the software application behaves as specified in the requirement document. Testing all the functionalities by providing appropriate input to verify whether the actual output is matching the expected output or not. It falls within the scope of black box testing and the testers need not concern about the source code of the application.

39. What is Non-Functional Testing?

In simple words, how well the system performs is non-functionality testing. Non-functional testing refers to various aspects of the software such as performance, load, stress, scalability, security, compatibility etc., Main focus is to improve the user experience on how fast the system responds to a request.

40. What is Acceptance Testing?

It is also known as pre-production testing. This is done by the end-users along with the testers to validate the functionality of the application. After successful acceptance testing. Formal testing conducted to determine whether an application is developed as per the requirement. It allows the customer to accept or reject the application. Types of acceptance testing are Alpha, Beta & Gamma.

41. On what basis is the acceptance plan prepared?

The acceptance test plan is prepared using the following inputs.

- Requirement Document: The requirement document specifies what exactly is needed and not needed in the existing project from the customer's perspective.
- Input from customer: Input from the customer will be in the format of formal emails, informal talks, discussions, etc.,
- Project plan: Project plan document prepared by the project manager.
- All the above three inputs act as good inputs to prepare the acceptance test plan.

42. What is Alpha Testing?

Alpha testing is done by the in-house developers (who developed the software) and testers before we ship the software to the customers. Sometimes alpha testing is done by the client or outsourcing team with the presence of developers or testers. It is a part of User Acceptance Testing. The purpose of doing this is to find bugs before the customers start using the software.

43. What is Beta Testing?

Beta testing is done by a limited number of end-users before delivery. It is done after the Alpha Testing. Usually, it is done in the client's place. Learn more about Beta Testing here.

44. What is Gamma Testing?

Gamma testing is done when the software is ready for release with specified requirements. It is done at the client place. It is done directly by skipping all the in-house testing activities.

45. What is Smoke Testing?

Smoke Testing is done to make sure if the build we received from the development team is testable or not. It is also called as “Day 0” check. It is done at the “build level”. It helps not to waste the testing time to simply testing the whole application when the key features don’t work or the key bugs have not been fixed yet.

46. What is Sanity Testing?

Sanity Testing is done during the release phase to check for the main functionalities of the application without going deeper. It is also called as a subset of Regression testing. It is done at the “release level”. At times due to release time constraints rigorous regression testing can’t be done to the build, sanity testing does that part by checking main functionalities.

47. What is Retesting?

To ensure that the defects which were found and posted in the earlier build were fixed or not in the current build. Say, Build 1.0 was released. Test team found some defects (Defect Id 1.0.1, 1.0.2) and posted. Build 1.1 was released, now testing the defects 1.0.1 and 1.0.2 in this build is retesting.

48. What is Regression Testing?

Repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the changes in the software being tested or in another related or unrelated software components.

Usually, we do regression testing in the following cases:

- New functionalities are added to the application
- Change Requirement (In organizations, we call it as CR)
- Defect Fixing
- Performance Issue Fix
- Environment change (E.g., Updating the DB from MySQL to Oracle)
- Read a detailed guide on Regression Testing

49. What do you mean by regression and confirmation testing?

Regression Testing: Testing team re-execute the tests against the modified application to make sure whether the modified code breaks anything which was working earlier.

Confirmation Testing: Usually testers report a bug when a test fails. Dev Team releases a new version of the software after the defect is fixed. Now the testing team will retest to make sure the reported bug is actually fixed or not.

50. What is Bug Life Cycle?

Bug life cycle is also known as Defect life cycle. In Software Development process, the bug has a life cycle. The bug should go through the life cycle to be closed. Bug life cycle varies depends upon the tools (QC, JIRA etc.,) used and the process followed in the organization. Click here for more details.

51. What is Bug Leakage?

A bug which is actually missed by the testing team while testing and the build was released to the Production. If now that bug (which was missed by the testing team) was found by the end user or customer, then we call it as Bug Leakage.

52. What is Bug Release?

Releasing the software to the Production with the known bugs then we call it as Bug Release. These known bugs should be included in the release note.

53. What is Defect Age?

Defect age can be defined as the time interval between date of defect detection and date of defect closure.

$$\text{Defect Age} = \text{Date of defect closure} - \text{Date of defect detection}$$

Assume, a tester found a bug and reported it on 1 Jan 2016 and it was successfully fixed on 5 Jan 2016. So the defect age is 5 days.

54. What is GUI Testing?

Graphical User Interface Testing is to test the interface between the application and the end user.

55. What is Recovery Testing?

Recovery testing is performed in order to determine how quickly the system can recover after the system crash or hardware failure. It comes under the type of non-functional testing.

56. What is Globalization Testing?

Globalization is a process of designing a software application so that it can be adapted to various languages and regions without any changes.

57. What is Random testing?

In random testing is a form of black-box software testing technique where the application is testing by generating random data.

58. What is Localization Testing (L10N Testing)?

Localization is a process of adapting globalization software for a specific region or language by adding local specific components.

59. What is Installation Testing?

It is to check whether the application is successfully installed and it is working as expected after installation.

60. What is Formal Testing?

It is a process where the testers test the application by having pre-planned procedures and proper documentation.

61. What is Risk Based Testing?

Identify the modules or functionalities which are most likely cause failures and then testing those functionalities.

62. What is Compatibility Testing?

It is to deploy and check whether the application is working as expected in a different combination of environmental components.

63. What is Exploratory Testing?

Usually, this process will be carried out by domain experts. They perform testing just by exploring the functionalities of the application without having the knowledge of the requirements. Check our detailed guide on Exploratory Testing and also don't miss these popular Exploratory Testing Tools.



64. What is Monkey Testing?

Perform abnormal action on the application deliberately in order to verify the stability of the application. Check our in-depth guide on Monkey Testing.

65. What is Usability Testing?

To verify whether the application is user-friendly or not and was comfortably used by an end-user or not. The main focus in this testing is to check whether the end-user can understand and operate the application easily or not. An application should be self-exploratory and must not require training to operate it. Check this guide to learn how to perform Usability Testing.

66. What is Security Testing?

Security testing is a process to determine whether the system protects data and maintains functionality as intended.

67. What is Soak Testing?

Running a system at high load for a prolonged period of time to identify the performance problems is called Soak Testing.

68. What is Endurance Testing?

Endurance testing is a non-functional testing type. It is also known as Soak Testing. Refer Soak testing.

69. What is Performance Testing?

This type of testing determines or validates the speed, scalability, and/or stability characteristics of the system or application under test. Performance is concerned with achieving response times, throughput, and resource-utilization levels that meet the performance objectives for the project or product.

70. What is Load Testing?

It is to verify that the system/application can handle the expected number of transactions and to verify the system/application behaviour under both normal and peak load conditions.

71. What is Volume Testing?

It is to verify that the system/application can handle a large amount of data

72. What is Stress Testing?

It is to verify the behaviour of the system once the load increases more than its design expectations.

73. What is Scalability Testing?

Scalability testing is a type of non-functional testing. It is to determine how the application under test scales with increasing workload.

74. What is Concurrency Testing?

Concurrency testing means accessing the application at the same time by multiple users to ensure the stability of the system. This is mainly used to identify deadlock issues.

75. What is Fuzz Testing?

Fuzz testing is used to identify coding errors and security loopholes in an application. By inputting a massive amount of random data to the system in an attempt to make it crash to identify if anything breaks in the application.



76. What is Adhoc Testing?

Ad-hoc testing is quite opposite to the formal testing. It is an informal testing type. In Adhoc testing, testers randomly test the application without following any documents and test design techniques. This testing is primarily performed if the knowledge of testers in the application under test is very high. Testers randomly test the application without any test cases or any business requirement document.

77. What is Interface Testing?

Interface testing is performed to evaluate whether two intended modules pass data and communicate correctly to one another.

78. What is Reliability Testing?

Perform testing on the application continuously for long period of time in order to verify the stability of the application

79. What is Bucket Testing?

Bucket testing is a method to compare two versions of an application against each other to determine which one performs better.

80. What is STLC?

STLC (Software Testing Life Cycle) identifies what test activities to carry out and when to accomplish those test activities. Even though testing differs between Organizations, there is a testing life cycle. Click here for more details.

81. What is RTM?

Requirements Traceability Matrix (RTM) is used to trace the requirements to the tests that are needed to verify whether the requirements are fulfilled. Requirement Traceability Matrix AKA Traceability Matrix or Cross Reference Matrix. Click here for more details.

82. What are the principles of Software Testing?

- Testing shows presence of defects
- Exhaustive testing is impossible
- Early testing
- Defect clustering
- Pesticide Paradox
- Testing is context depending
- Absence of error fallacy
- Click here for more details.

83. What is Exhaustive Testing?

Testing all the functionalities using all valid and invalid inputs and preconditions is known as Exhaustive testing.

84. What is Early Testing?

Defects detected in early phases of SDLC are less expensive to fix. So conducting early testing reduces the cost of fixing defects.

85. What is Defect clustering?

Defect clustering in software testing means that a small module or functionality contains most of the bugs or it has the most operational failures.

86. What is Pesticide Paradox?

Pesticide Paradox in software testing is the process of repeating the same test cases, again and again, eventually, the same test cases will no longer find new bugs. So to overcome this Pesticide Paradox, it is necessary to review the test cases regularly and add or update them to find more defects.

87. What is Defect Cascading in Software Testing?

Defect cascading in Software testing means triggering of other defects in an application. When a defect is not identified or goes unnoticed while testing, it invokes other defects. It leads to multiple defects in the later stages and results in an increase in a number of defects in the application.

For example, if there is a defect in an accounting system related to negative taxation then the negative taxation defect affects the ledger which in turn affects other reports such as Balance Sheet, Profit & Loss etc.,

88. What is Walk Through?

A walkthrough is an informal meeting conducted to learn, gain understanding, and find defects. The author leads the meeting and clarifies the queries raised by the peers in the meeting.

89. What is HotFix?

A bug that needs to be handled as a high priority bug and fixed immediately.

90. What is Inspection?

Inspection is a formal meeting led by a trained moderator, certainly not by the author. The document under inspection is prepared and checked thoroughly by the reviewers before the meeting. In the inspection meeting, the defects found are logged and shared with the author for appropriate actions. Post inspection, a formal follow-up process is used to ensure a timely and corrective action.

91. Who are all involved in an inspection meeting?

Author, Moderator, Reviewer(s), Scribe/Recorder and Manager.

92. What is a Defect?

The variation between the actual results and expected results is known as a defect. If a developer finds an issue and corrects it by himself in the development phase, then it's called a defect. [Click here for more details.](#)

93. What is a Bug?

If testers find any mismatch in the application/system in testing phase, then they call it as Bug. [Click here for more details.](#)

94. What is an Error?

We can't compile or run a program due to a coding mistake in a program. If a developer is unable to successfully compile or run a program, then they call it as an error. [Click here for more details.](#)

95. What is a Failure?

Once the product is deployed and customers find any issues then they call the product as a failure product. After release, if an end user finds an issue then that particular issue is called as a failure. [Click here for more details.](#)



96. What is Bug Severity?

Bug/Defect severity can be defined as the impact of the bug on customer's business. It can be Critical, Major or Minor. In simple words, how much effect will be there on the system because of a particular defect. Click [here](#) for more details.

97. What is Bug Priority?

Defect priority can be defined as how soon the defect should be fixed. It gives the order in which a defect should be resolved. Developers decide which defect they should take up next based on the priority. It can be High, Medium or Low. Most of the times the priority status is set based on the customer requirement. Click [here](#) for more details.

98. Tell some examples of Bug Severity and Bug Priority?

High Priority & High Severity: Submit button is not working on a login page and customers are unable to login to the application

Low Priority & High Severity: Crash in some functionality which is going to deliver after couple of releases

High Priority & Low Severity: Spelling mistake of a company name on the homepage

Low Priority & Low Severity: FAQ page takes a long time to load

99. What is a Critical Bug?

A critical bug is a show stopper which means a large piece of functionality or major system component is completely broken and there is no workaround to move further.

For example, due to a bug in one module, we cannot test the other modules because that blocker bug has blocked other modules. Bugs which affects the customers' business are considered as critical.

Example:

1. "Sign In" button is not working on Gmail App and Gmail users are blocked to login to their accounts.
2. An error message pops up when a customer clicks on transfer money button in a Banking website.

100. What is the difference between a Standalone application, Client-Server application and Web application?

- **Standalone application:** Standalone applications follow one-tier architecture. Presentation, Business, and Database layer are in one system for a single user.
- **Client-Server Application:** Client-server applications follow two-tier architecture. Presentation and Business layer are in a client system and Database layer on another server. It works majorly in Intranet.
- **Web Application:** Web server applications follow three-tier or n-tier architecture. The presentation layer is in a client system, a Business layer is in an application server and Database layer is in a Database server. It works both in Intranet and Internet.

101. What is State Transition?

Using state transition testing, we pick test cases from an application where we need to test different system transitions. We can apply this when an application gives a different output for the same input, depending on what has happened in the earlier state. Click [here](#) for more details.

102. What is entry criteria?

The prerequisites that must be achieved before commencing the testing process. Click [here](#) for more details.

103. What is exit criteria?

The conditions that must be met before testing should be concluded. Click [here](#) for more details.

104. What is SDLC?

Software Development Life Cycle (SDLC) aims to produce a high-quality system that meets or exceeds customer expectations, works effectively and efficiently in the current and planned information technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.

105. What is Error Seeding?

Error seeding is a process of adding known errors intendedly in a program to identify the rate of error detection. It helps in the process of estimating the tester skills of finding bugs and also to know the ability of the application (how well the application is working when it has errors.)

106. What is Error Guessing?

Error guessing is also a method of test case design similar to error seeding. In error guessing, testers design test cases by guessing the possible errors that might occur in the software application. The intention is to catch the errors immediately.

107. What is Showstopper Defect?

A showstopper defect is a defect which won't allow a user to move further in the application. It's almost like a crash.

Assume that login button is not working. Even though you have a valid username and valid password, you could not move further because the login button is not functioning.

108. Can you do System testing at any stage of SDLC?

We can do System Testing only when all the units are in place and working properly. It can only be done before User Acceptance Testing (UAT).

109. What are the different strategies for rollout to end-users?

There are four strategies to be followed for the rollout of any software testing project are as follows:

- Pilot
- Gradual Implementation
- Phased Implementation
- Parallel Implementation

110. What is Boundary Value Analysis?

Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions. The Behaviour at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects. Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition. A boundary value for a valid partition is a valid boundary value. Similarly, a boundary value for an invalid partition is an invalid boundary value. [Click here for more details.](#)

111. What is Equivalence Class Partition?

Equivalence Partitioning is also known as Equivalence Class Partitioning. In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be proposed in the same way. Hence selecting one input from each group to design the test cases. [Click here for more details.](#)



112. What is Decision Table testing?

Decision Table is aka Cause-Effect Table. This test technique is appropriate for functionalities which has logical relationships between inputs (if-else logic). In the Decision table technique, we deal with combinations of inputs. To identify the test cases with a decision table, we consider conditions and actions. We take conditions as inputs and actions as outputs. Click here for more details.

113. What are the different available models of SDLC?

1. Waterfall
2. Spiral
3. V Model
4. Prototype
5. Agile

114. What is the procedure of manual testing?

Manual testing is crucial for testing software applications more thoroughly. The procedure of manual testing comprises of the following.

1. Planning and Control
2. Analysis and Design
3. Implementation and Execution
4. Evaluating and Reporting
5. Test Closure activities

115. What is Test Metrics?

Software test metrics is to monitor and control process and product. It helps to drive the project towards our planned goals without deviation. Metrics answer different questions. It's important to decide what questions you want answers to. Click here for more details.

116. What is API Testing?

API testing is a type of software testing that involves testing APIs directly and also as a part of integration testing to check whether the API meets expectations in terms of functionality, reliability, performance, and security of an application. In API Testing our main focus will be on a Business logic layer of the software architecture. API testing can be performed on any software system which contains multiple APIs. API testing won't concentrate on the look and feel of the application. API testing is entirely different from GUI Testing.

117. Which test cases are written first white boxes or black box?

The simple answer is black-box test cases are written first.

Let's see why black-box test cases are written first compared to white box test cases.

Prerequisites to start writing black-box test cases are Requirement documents or design documents. These documents will be available before initiating a project.

Prerequisites to start writing white box test cases are the internal architecture of the application. The internal architecture of the application will be available in the later part of the project i.e., designing.