10

Unit - VI

SOFTWARE TESTING

Syllabus

Strategic Approach to Software Testing, Verification and Validation, Organizing for Software Testing, Software Testing Strategy - The Big Picture, Criteria for Completion of Testing, Strategic Issues, Test Strategies for Conventional Software, Unit Testing, Integration Testing, Test Strategies for Object-Oriented Software, Unit Testing in the OO Context, Integration Testing in the OO Context, Test Strategies for WebApps, Validation Testing, Validation-Test Criteria, Configuration Review.

Suggested Free Open Source Tools : Selenium, JUnit.

10.1 A Strategic Approach to Software Testing

University Question

Q. What is software testing? Explain the software testing strategies for software development?

SPPU - May 18, May 19, 7 Marks

- Testing is a set of pre-planned activities that can be conducted systematically. A number of software testing strategies have been proposed in the literature. All provide the software developer with a template for testing and all have the following generic characteristics :
  - To perform effective testing, a software team should conduct effective formal technical reviews. By doing this, many errors will be eliminated before testing commences.
  - Testing begins at the component level and works "outward" towards the integration of the entire computer-based system.
  - Different testing techniques are appropriate at different points in time.
  - Testing is conducted by the developer of the software and (for large projects) an independent test group.
  - Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.
- A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high-level tests that validate major system functions against customer requirements.

10.1.1 Verification and Validation

- Software testing is one element of a Verification and Validation (V&V). Verification refers to the set of activities that ensure that software correctly implements a specific function.

- Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.
- The definition of V&V encompasses many of the activities that are encompassed by Software Quality Assurance (SQA).
- SQA activities include formal technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility study, documentation review, database review, algorithm analysis, development testing, usability testing, qualification testing, and installation testing.

10.1.2 Organizing for Software Testing

- For every software project, there is an inherent conflict of interest that occurs as testing begins. The people who have built the software are now asked to test the software.
- This seems harmless in itself; after all, who knows the program better than its developers?
- Unfortunately, these same developers have a vested interest in demonstrating that the program is error free, that it works according to customer requirements, and that it will be completed on schedule and within budget. Each of these interests mitigate against thorough testing.
- From a psychological point of view, software analysis and design (along with coding) are constructive tasks. The software engineer analyzes, models, and then creates a computer program and its documentation.
- From the point of view of the builder, testing can be considered to be (psychologically) destructive.
- So the builder treads lightly, designing and executing tests that will demonstrate that the program works, rather than uncovering errors. Unfortunately, errors will be present.
- There are often a number of misconceptions those can be erroneously inferred from the preceding discussion :
  1) That the developer of software should do no testing at all,
  2) That the software should be "tossed over the wall" to strangers who will test it mercilessly,
  3) That testers get involved with the project only when the testing steps are about to begin. Each of these statements is incorrect.
- The software developer is always responsible for testing the individual units (components) of the program.
- Only after the software architecture is completed, one Independent Test Group (ITG) is involved.
- The role of an Independent Test Group (ITG) is to remove the inherent problems associated with letting the builder test the thing that has been built.
- Independent testing removes the conflict of interest that may otherwise be present. After all, ITG personnel are paid to find errors.
- The ITG is part of the software development project team in the sense that it becomes involved during analysis and design and stays involved (planning and specifying test procedures) throughout a large project.

10.1.3 Software Testing Strategy

- The software process may be viewed as the spiral illustrated in Fig. 10.1.1.

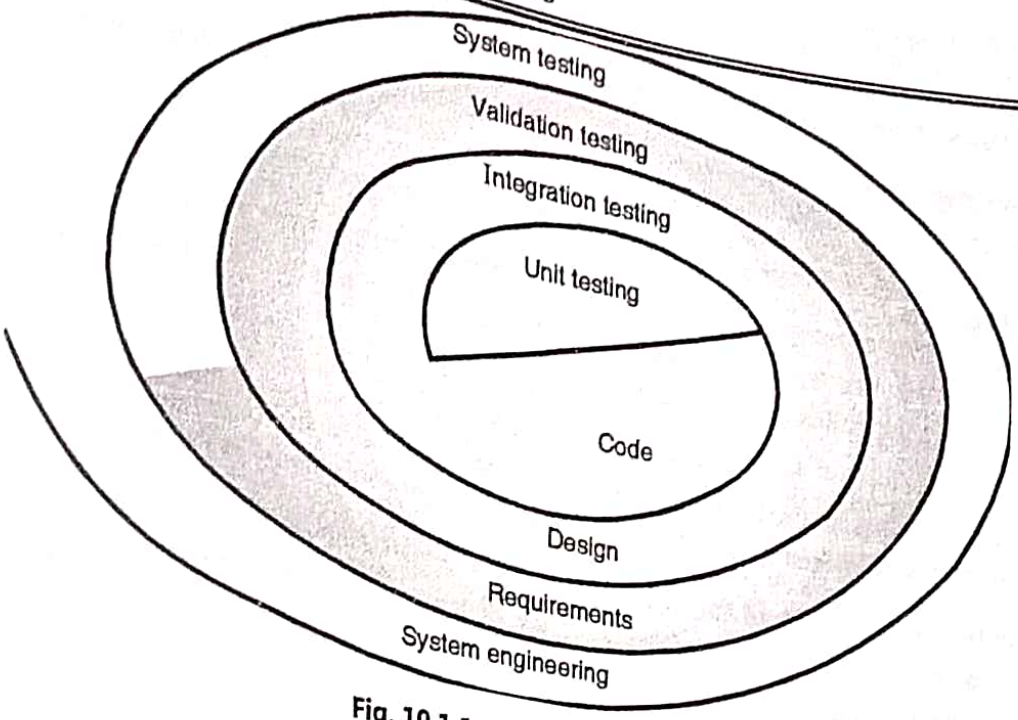

Fig. 10.1.1 : Testing strategy

- Initially, system engineering defines the role of software and leads to software requirements analysis, where the information domain, function, behaviour, performance, constraints, and validation criteria for software are established.
- A strategy for software testing may also be viewed in the context of the spiral.
- **Unit testing** begins at the vortex of the spiral and concentrates on each unit of the software as implemented in source code.
- Testing progresses by moving outward along the spiral to **integration testing**, where the focus is on design and the construction of the software architecture.
- Taking another turn outward on the spiral, we encounter **validation testing**, where requirements established as part of software requirements analysis are validated against the software that has been constructed.
- Finally, we arrive at **system testing**, where the software and other system elements are tested as a whole.
- To test computer software, we spiral out along streamlines that broaden the scope of testing with each turn.
- Considering the process from a procedural point of view, testing within the context of software engineering is actually a series of four steps that are implemented sequentially. The steps are shown in Fig. 10.1.2.

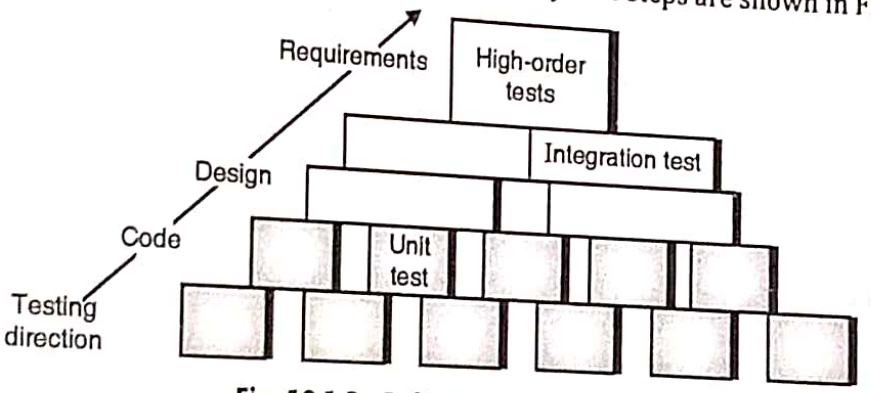

Fig. 10.1.2 : Software testing steps

1. **Unit testing :** Initially, tests focus on each component individually, ensuring that it functions properly as a unit.
2. **Integration testing :** Integration testing addresses the issues associated with the dual problems of verification and program construction. Test case design techniques that focus on inputs and outputs are more prevalent during integration.

3. **High-Order tests :** After the software has been integrated (constructed), sets of high-order tests are conducted.
4. **Validation testing :** Validation testing provides final assurance that software meets all functional, behavioural, and performance requirements.

10.2 The Big Picture

- Software engineering has become a very important part of computer science, so much so that today's programmers are rarely called programmers, but instead called software engineers.
- As software becomes larger, the teams working on it have grown, and good communication skills have become even more important than in the past. Furthermore, computer systems are only useful if they make things better for humans, so developers need to be good at understanding the users they are developing software for.
- As computers become smaller and cheaper, we have gone from having shared computers that humans have to queue up to use, to having multiple digital devices for each person. Now, it's the devices that have to wait until the human is ready. In a digital system, the human is the most important part.
- **"Big picture" is a relative term :**
  - For the junior software engineer, writing code correctly and quickly is part of the big picture of effective project delivery.
  - For the senior associate, it is about contributing to the overall project with his team.
  - For the manager, it is about contributing to the client.
  - For the company, it is about building good business relationships based on mutual trust and co-operation.
  - For the clients, it is about giving value to their own customers.

10.3 Criteria for Completion of Testing

- The completion criteria are the conditions that are specified for stopping the testing. Sometimes it becomes very difficult to say when the testing phase is completed and when the product can be released.
- The stop-test criteria are as follows :
  - All the planned testing have been executed and completed successfully. Also it has been passed.
  - All the specified goals have been achieved.
  - All the defects have been removed and all the errors have been uncovered and resolved successfully.
  - Detection of defects have been completed within stipulated time.
  - The project time has run out.
  - When we are not able to detect any known serious defects
  - When the specified coverage has been successfully attained.

10.4 Strategic Issues

Following issues must be addressed if a successful software testing strategy is to be implemented :

- **Specify product requirements in a quantifiable manner long before testing commences**

Although the overriding objective of testing is to find errors, a good testing strategy also assesses other quality characteristics such as portability, maintainability, and usability. These should be specified in a way that is measurable so that testing results are unambiguous.

State testing objectives explicitly

- The specific objectives of testing should be stated in measurable terms. For example, test effectiveness, test coverage, mean time to failure, the cost to find and fix defects, remaining defect density or frequency of occurrence, and test work-hours per regression test all should be stated within the test plan.

Understand the users of the software and develop a profile for each user category

- Use-cases that describe the interaction scenario for each class of user can reduce overall testing effort by focusing testing on actual use of the product.

Develop a testing plan that emphasizes "rapid cycle testing"

- The feedback generated from these rapid cycle tests can be used to control quality levels and the corresponding test strategies.

Build "robust" software that is designed to test itself

- Software should be designed in a manner that uses antibugging techniques. That is, software should be capable of diagnosing certain classes of errors. In addition, the design should accommodate automated testing and regression testing.

Use effective formal technical reviews as a filter prior to testing

- Formal technical reviews can be as effective as testing in uncovering errors. For this reason, reviews can reduce the amount of testing effort that is required to produce high-quality software.

Conduct formal technical reviews to assess the test strategy and test cases themselves

- Formal technical reviews can uncover inconsistencies, omissions, and outright errors in the testing approach. This saves time and also improves product quality.

Develop a continuous improvement approach for the testing process

- The test strategy should be measured. The metrics collected during testing should be used as part of a statistical process control approach for software testing.

10.5 Test Strategies for Conventional Software

- For the conventional software, there are various test strategies available for use. In first one, the team waits for a product development to be completed and then start conducting tests.
- In another approach, the developer conducts the test on a daily basis as the development proceeds and one part of the program is completed.
- These strategies are chosen by the development team based on the product and convenience of the team. The choice may among two approaches discussed above.
- The team starts with an incremental view of testing and test individual program units. Then moving ahead with integration of the units, and finalizing the tests of overall system.

Different types of tests are mentioned as follows :

1. Unit Testing
2. Integration Testing
3. Regression Testing
4. Smoke Testing

10.5.1 Unit Testing

University Question

Q. Explain unit testing. **SPPU - May 12, 4 Marks**

In unit testing, the main focus is on assessment of smallest unit of the product design like component of the software or module. The unit testing has limited scope and it emphasizes on internal processing logic and data structures. Multiple modules and components can be tested in parallel.

Unit test considerations

- The unit testing is illustrated diagrammatically as shown in Fig. 10.5.1.

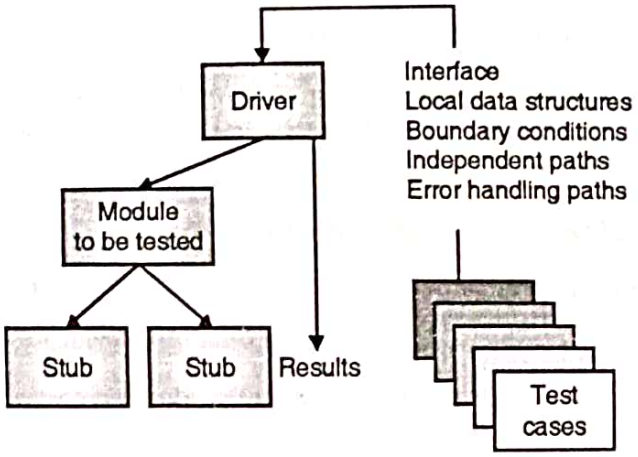

Fig. 10.5.1 : Unit test

- The test strategy is conducted on each of the **module interface** to assess the proper flow of input and output and its correctness as per the requirements.
- Next, the **local data structures** are assessed to ensure its integrity in the execution of the algorithm.
- All the **independent paths** (also called basis paths) are also evaluated through the control structure and it keeps track on the program and makes sure that at least each of the statements in a module should have been executed once.
- **Boundary conditions** are also tested so that the software works properly within the boundaries or within the limits. All the error **handling paths** are tested.
- Thus the **boundary testing** is one of the significant unit testing methodology.

Unit test procedures

- In parallel with coding step, usually unit testing is conducted. Before the start of coding, unit test can be conducted. This method is most commonly used in agile development process.
- The design information gives sufficient help for the developers to establish the test cases and uncover the errors. The developers always couple the set of results with the test cases.

Unit test environment

- The unit test environment is illustrated in Fig. 10.5.2.

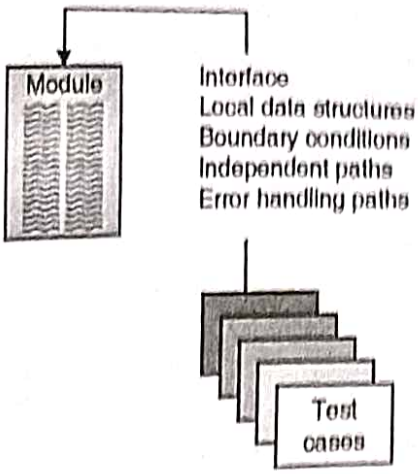

Fig. 10.5.2 : Unit test environment

- In most of the applications, a driver is considered as the "main program". This main program itself accepts all the test case data and pass that data to the component under test and the results are also printed side by side.
- In the Fig. 10.5.2, we observe that the stubs replace the modules of the program and are tested next to driver. The stub is also considered to be the subprogram. These subprograms make an interface with the main program to complete the test.
- Logically both the driver and stubs are the software that are written but not submitted to the customer and thus are considered as the overhead. It is always recommended to keep these overhead simple to reduce the cost. But overhead can not be made simple in all the cases and hence the unit testing can be postponed to the integration testing.

10.5.2 Integration Testing

University Question

Q. What do you understand by Integration Testing ? Explain objectives of Integration Testing. **SPPU - Dec. 19, 6 Marks**

- Integration testing is used for constructing the software architecture. It is a systematic approach for conducting tests to uncover interfacing errors. The main objective behind integration testing is to accept all the unit tested components and integrate into a program structure as given by the design.
- It is often observed that there is a tendency to attempt always non-incremental approaches. All the components are integrated in the beginning and the program is tested as a whole.
- In this approach a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Small increments of the program are tested in an incremental approach. In incremental integration approach, the error detection and correction is quite simple. In this all the interfaces are tested completely and thus a systematic test strategy may be applied.
- Following are different incremental integration strategies :

1) Top-down integration

- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.

- All the modules are combined by moving top to down direction through the control hierarchy. It begins with the main program or the main control module. The flow is from main module to its subordinate modules in depth-first or breadth-first manner.
- Depth-first integration is illustrated in Fig. 10.5.3, and it integrates all the components on most of the control path of the program.

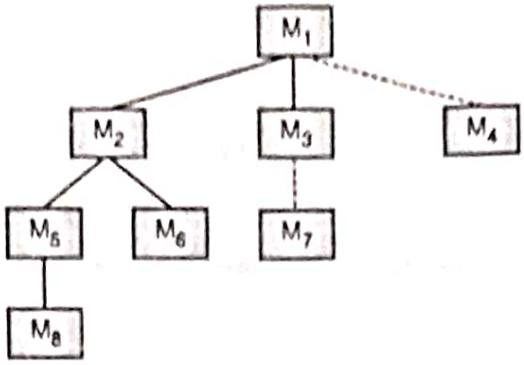

Fig. 10.5.3 : Top-down integration

2) Bottom-up integration

- In bottom-up integration testing, the testing begins with the construction and components at the lowest level. These components are called as atomic modules.
- Since the components are integrated from the bottom to top, the required processing is always available for components subordinate in all the levels. Here in this approach the need of stub is completed eliminated. The use of stub was actually a disadvantage due to overhead.
- Following steps are required for implementation of bottom-up integration strategy :
- The low-level components are integrated to form clusters that can perform sub function of a specific software.
- A driver is needed to coordinate all the test case inputs and outputs.
- Later all the clusters are tested.
- Then, drivers are removed and all the clusters are integrated once again from bottom to top direction in the program.
- Integration follows the pattern illustrated in Fig. 10.5.4.

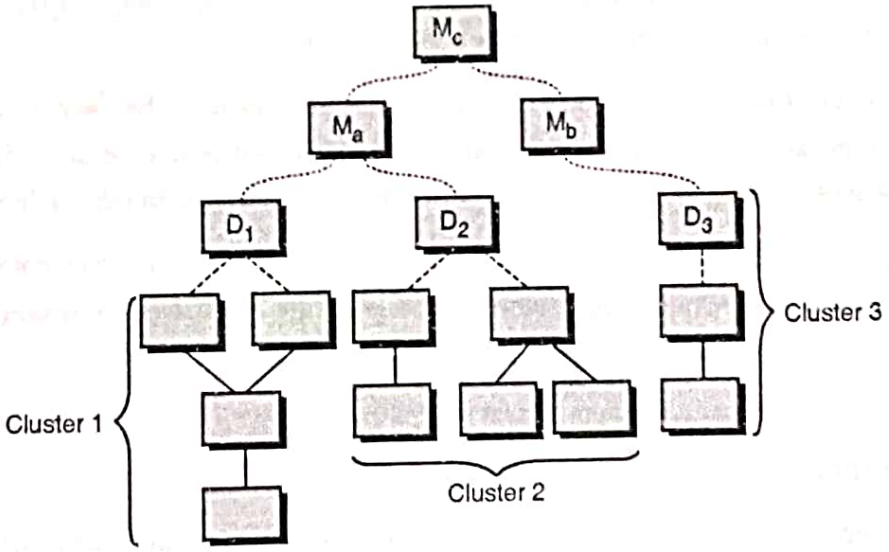

Fig. 10.5.4 : Bottom-up integration

- The components are integrated in such a fashion that they can form clusters 1, 2 and 3. These clusters are tested by using a driver (marked by a dashed block).
- The components in clusters 1 and 2 are subordinate to module M_a. The drivers D_1 and D_2 are then removed and the clusters are connected directly to M_a.
- In the same way, the driver D_3 for the cluster 3 is removed and then cluster 3 is directly connected to module M_b. Here both the modules, M_a and M_b are finally integrated with the component M_c, and so on.

Problems associated with Top down approach of testing

- In incremental integration testing approach, a set of errors are encountered and correction seems to be very difficult due to isolation of causes and the complications by program since program is expanded over several modules. After correction of these errors, some new may appear and the process continues. It looks like an infinite loop.
- Top-down integration testing is one of the incremental testing strategies for construction of the software architecture.
- The main problem with top down approach is that the test conditions are nearly impossible or they are extremely difficult to create.
- Stub modules are complicated.

10.6 Test Strategies for Object-Oriented Software

- The objective of any testing strategy is simply uncovering the maximum number of possible errors with optimum amount of efforts applied over a realistic time span.
- This basic idea remains same but the Object-Oriented (OO) software changes both testing strategy and testing tactics.

10.6.1 Unit Testing in OO Context

- Whenever the object-oriented software is tested, the concept of the unit testing changes as compared to conventional approaches. Encapsulation is preferred in object oriented software and concept of classes is used.
- Each of classes and their instances are called as objects. The package attributes are the data and operations are nothing but the functions in object oriented context. An encapsulated class is the main focus of unit testing in object oriented context.
- Operations in each of the classes are the smallest units that can be unit tested.
- In conventional software, we call it as unit testing and in case of object oriented software, it is referred as class testing.

10.6.2 Integration Testing in OO Context

- Since the object-oriented software does not have any fixed hierarchical control structure, the top-down and bottom-up integration strategies becomes meaningless.
- Also, the integrating operations one at a time into a class is impossible due to the direct and indirect interactions of the components of that class.

- Following are two different strategies for integration testing in OO context :

1. Thread-based testing
2. Use-based testing

- The **thread-based testing** combines the set of classes that are required to respond to the input for the system.
- The **use-based testing** start with the construction of the system and it tests those classes only that uses very few server classes.

10.7 Test Strategies for WebApps

- The testing strategy for Web App uses the basic principles for all software testing and also applies the strategies and tactics of object-oriented testing.
- Following steps summarize the approach :
  - The content model for Web App is reviewed to uncover errors.
  - The interface model is reviewed to ensure that all use cases can be accommodated.
  - The design model for Web App is reviewed to uncover navigation errors.
  - The user interface is tested to uncover errors in presentation or navigation mechanics.
  - Each functional component is unit tested.
  - Navigation throughout the architecture is tested.
  - The Web App is implemented in a variety of different environmental configurations.
  - Security tests are conducted.
  - Performance tests are conducted.
  - The Web App is tested by a controlled and monitored population of end users.

10.8 Validation Testing

- Validation can be defined in many ways, but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.
- Reasonable expectations are defined in the Software Requirements Specification.

10.8.1 Validation-Test Criteria

- Software validation is achieved through a series of tests that demonstrate conformity with requirements.
- A **test plan** outlines the classes of tests to be conducted, and a **test procedure** defines specific test cases.
- Both the plan and procedure are designed to ensure that all functional requirements are satisfied, all behavioral characteristics are achieved, all performance requirements are attained, documentation is correct, and usability and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability).
- After each validation test case has been conducted, one of two possible conditions exists :

  1) The function or performance characteristic conforms to specification and is accepted, or

  2) A deviation from specification is uncovered and a deficiency list is created.

- Deviation or error discovered at this stage in a project can rarely be corrected prior to scheduled delivery.
- It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

10.8.2 Configuration Review

- An important element of the validation process is a configuration review.
- The intent of the review is to ensure that all elements of the software configuration have been properly developed, are cataloged, and have the necessary detail to bolster the support phase of the software life cycle.

10.8.3 Alpha and Beta Testing

University Question

Q. Differentiate between alpha and beta testing. **SPPU - May 10, Dec. 10, 6 Marks**

- It is virtually impossible for a software developer to foresee how the customer will really use a program.
- Instructions for use may be misinterpreted, strange combinations of data may be regularly used; output that seemed clear to the tester may be unintelligible to a user in the field.
- When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements.
- Conducted by the end-user rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests.
- In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.
- If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one.
- Most software product builders use a process called alpha and beta testing to uncover errors that only the end-user seems able to find.
- The alpha test is conducted at the developer's site by end-users. The software is used in a natural setting with the developer "looking over the shoulder" of typical users and recording errors and usage problems.
- Alpha tests are conducted in a controlled environment.
- The beta test is conducted at end-user sites. Unlike alpha testing, the developer is generally not present.
- Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.
- The end-user records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.
- As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.

10.9 Case Study - Open Source Tool : Selenium

10.9.1 Introduction : Selenium

- Software testing is an indispensable part of every software development lifecycle and allows for improved quality of developed systems. For this reason, various test procedures have been developed to achieve the desired product quality. Nowadays, when new technologies, platforms, and programming languages are emerging, soft-ware testing is still developing and changing. Recent topics include mobile application testing, cloud testing, virtualization, or the use of automation tools.

- The objective of software testing is to discover faults and errors in a software application. Software testing utilizes more than 50 % time of software development lifecycle. Testing time depends upon the algorithm used, programming language, line of codes, function points, external and internal interfaces. Testing improves the quality of an application.
- To ease the testing process, there are various programmed web testing tools are available in the market, a detailed survey is given. As compared to human testing, automatic testing is better, because automation testing systematizes the testing process of analysis, design and coding of various scripts. Selenium is an open source tool which is used for test automation of different types of online and real-time applications.
- Since the Selenium was proposed to automate web browser interface, therefore programming scripts can spontaneously accomplish the similar interactions that some performed manually. Although the Selenium can achieve any type of automated interaction, but was planned and initially used for automated web application testing. It is licensed under Apache License 2.0 and portable automated software testing tool. It has proficiencies to function through different browsers and operating systems. It is not just a single tool but a set of tools that assists testers to automate web-based applications more professionally. The selenium can't test desktop based applications.

10.9.2 Architecture of the Selenium

- Following Fig. 10.9.1 describes the architecture of the Selenium web tool. It has two basic components: Selenium Client and Server. The client includes the WebDriver API, which is used to create test scripts to make interaction with web page and other application elements. It also includes the remote Web Driver class, which communicates with a remote Selenium server.

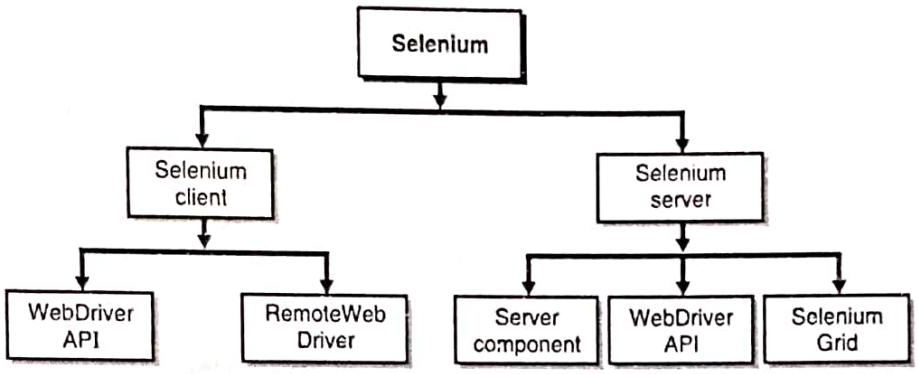

Fig. 10.9.1 : Selenium Architecture

- The Selenium server comprises of a server component which is used to accept requests from Selenium Client's Remote Web Driver class. It also comprises of the Web Driver API which is used to execute tests for web browsers on a server machine. The last component is the Selenium Grid which is implemented by Selenium Server in command-line options for grid characteristics, containing a central hub and nodes for numerous situations and preferred browser capabilities. Grid is a tool expended to run parallel tests through dissimilar machines and dissimilar browsers concurrently which effects the minimized execution time.

10.9.3 Selenium Integrated Development Environment (IDE)

- The IDE is a Firefox plugin that permits testers to record their actions as they track the work process flow that they want to test. The main components of the Selenium IDE are as follows. Selenium IDE has Menu bar, Base URL text box, Tool bar, Test Case pane, Command window and test Run or Failure status section.
- Fig. 10.9.2 shows the File menu which contains options for Test Case and Test Suite as redrawn form, Fig. 10.9.3 shows actions menu which permits users to record, play entire test suite, play current test case, play or pause, play fast or slow, set break point, and execute current test step.

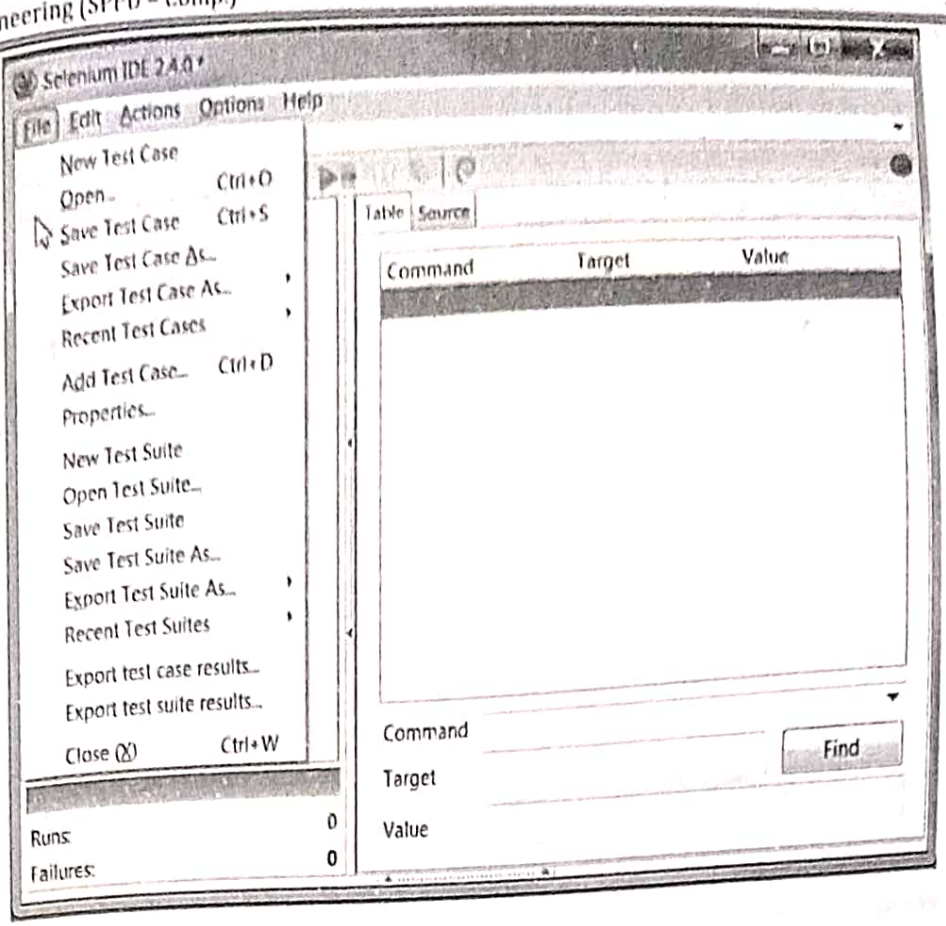

Fig. 10.9.2 : File Menu of Selenium

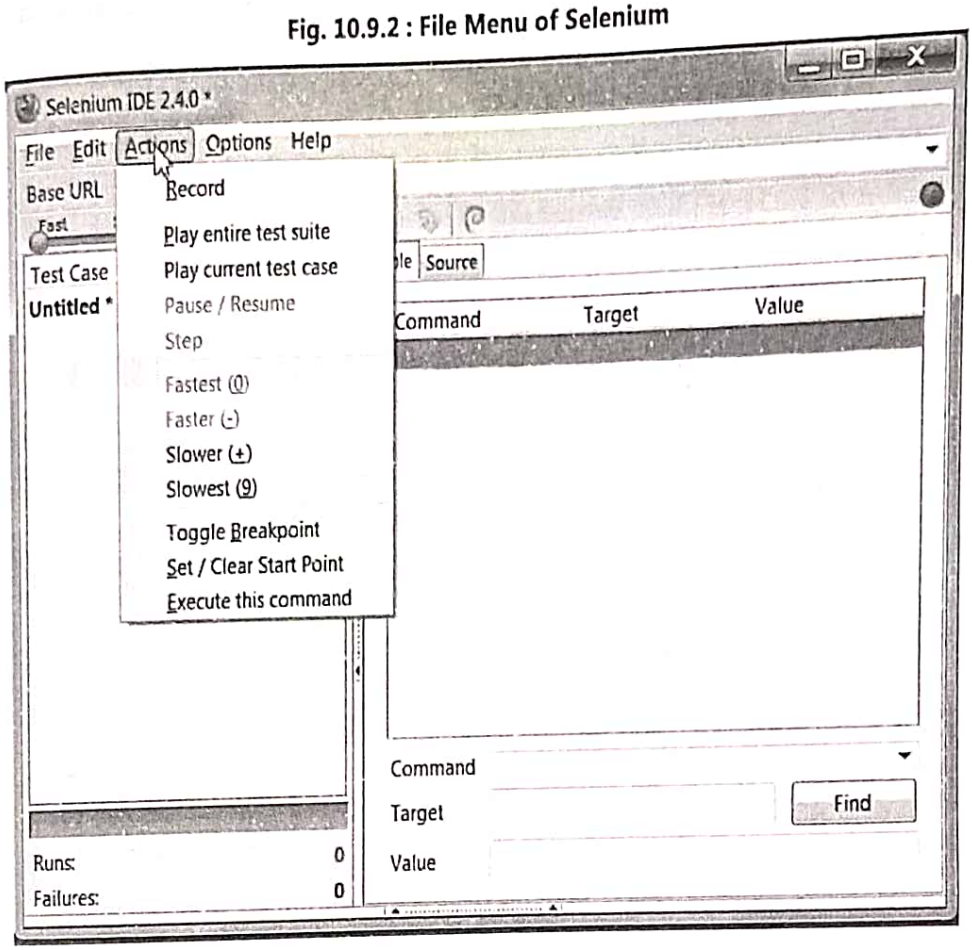

Fig. 10.9.3 : Action Menu of Selenium

- Fig. 10.9.4 shows the Help menu which permits testers to analysis documentation, UI-Element documentation and traverse to Selenium websites. Fig. 10.9.5 shows The Edit menu which permits the following options like copy, paste, delete, undo, and select all operations for editing the commands in the test case.

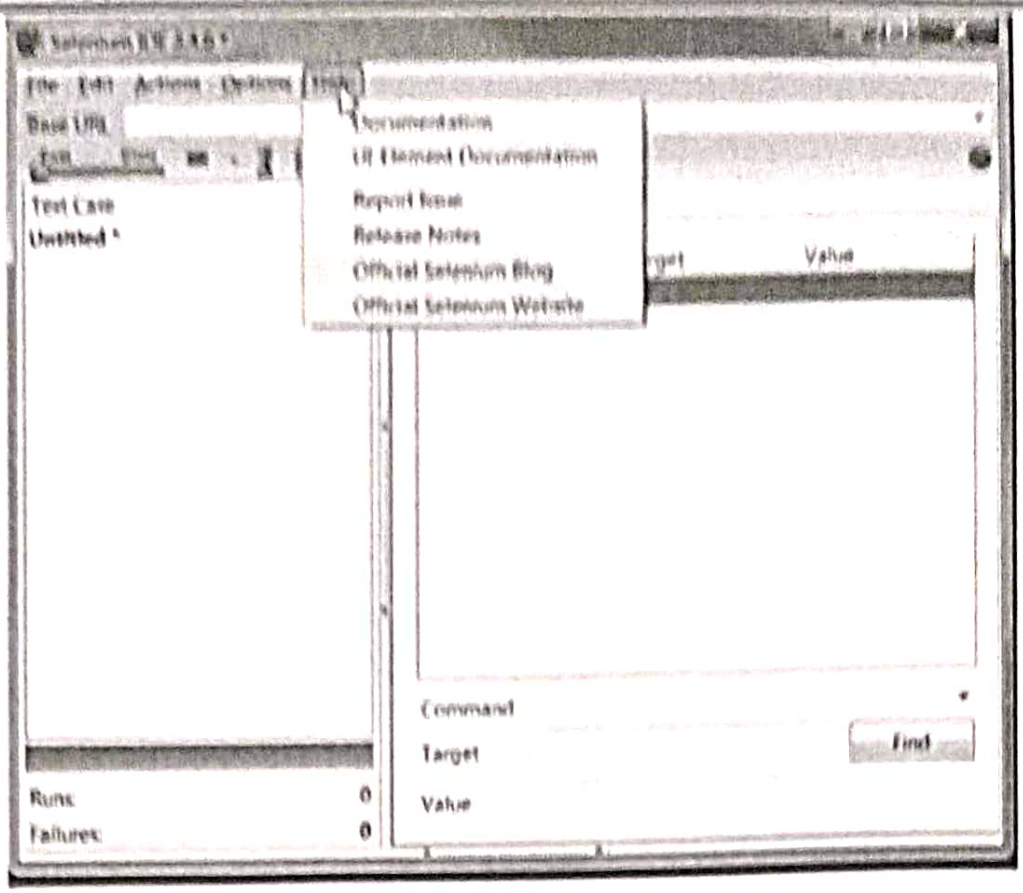

Fig. 10.9.4 : Help Menu of Selenium

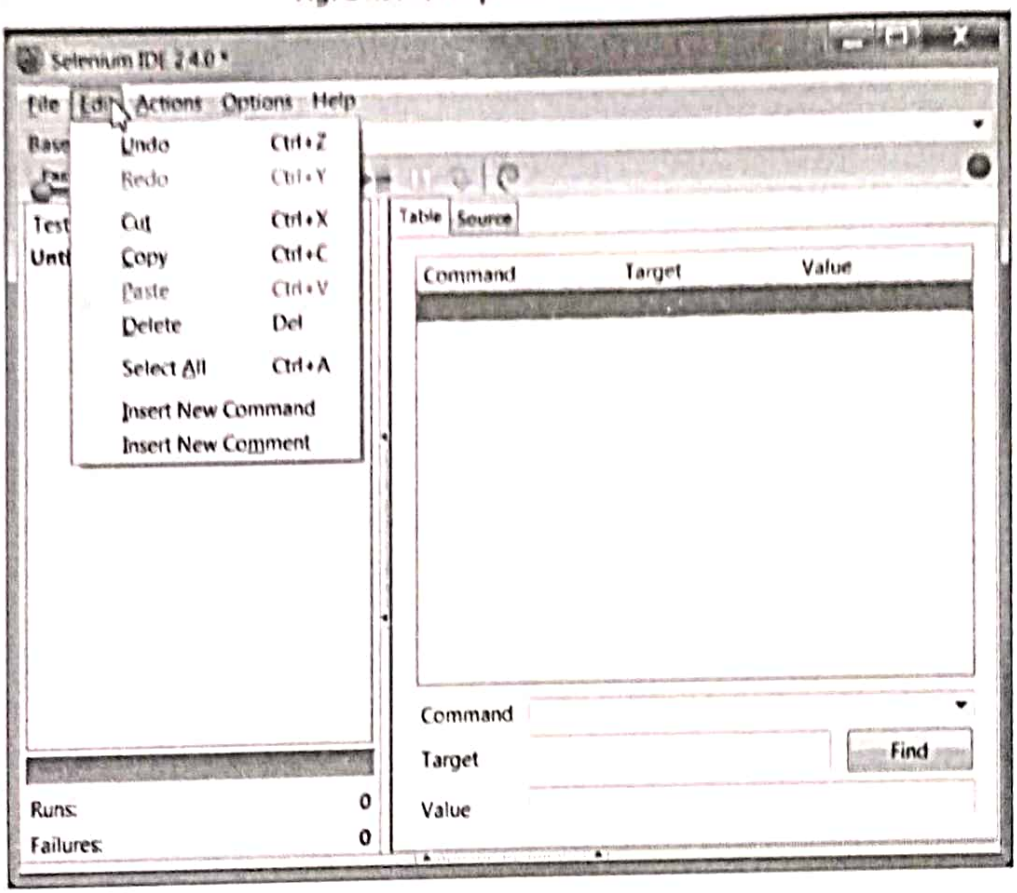

Fig. 10.9.5 : Edit Menu of Selenium

10.9.4 Testing Using Selenium

Step 1 : First of all search selenium IDE with the help of Mozilla Firefox page, after search; download the Add-on IDE using Firefox. Next restart the browser and see the Selenium IDE in browser as shown in Fig. 10.9.6.

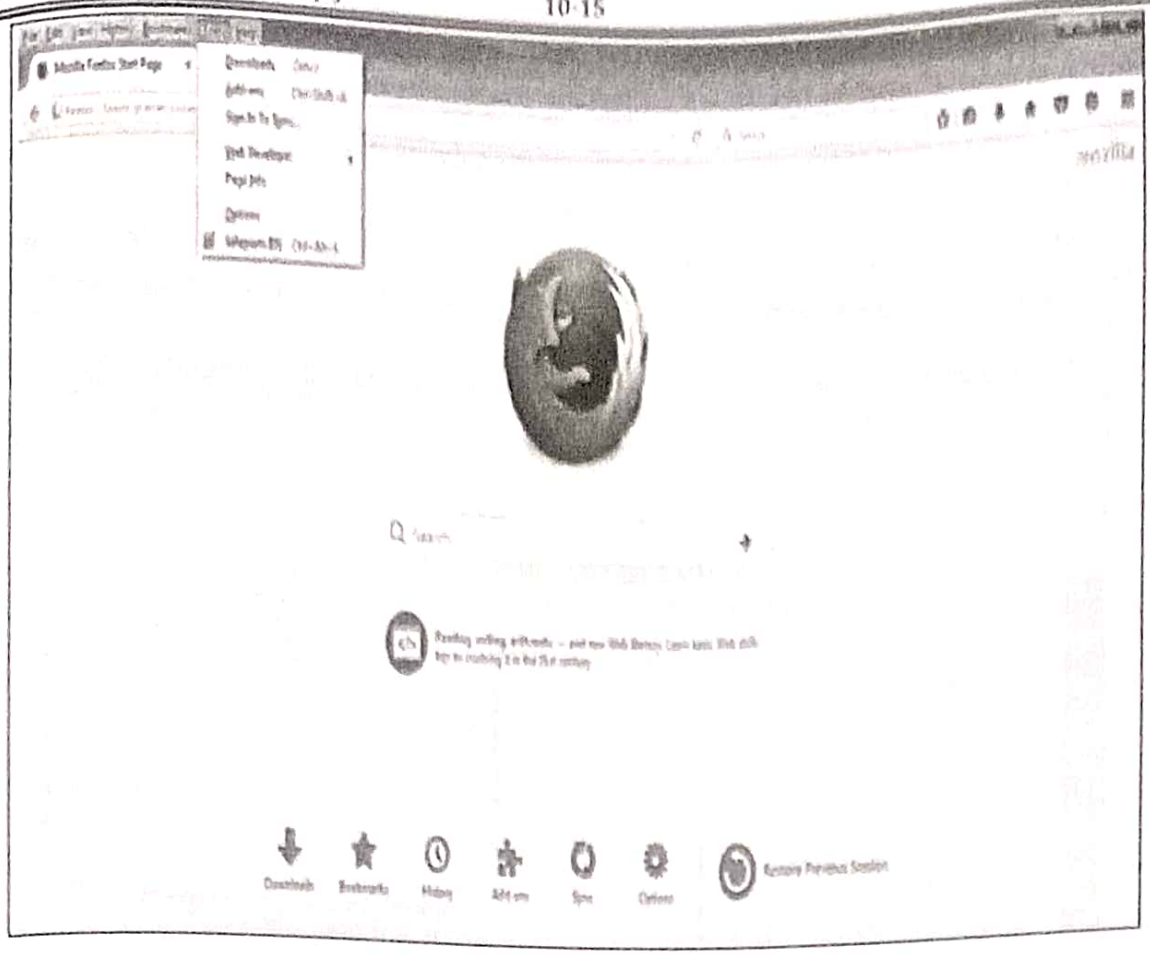

Fig. 10.9.6 : Selenium IDE in FireFox

2 : After click on Selenium IDE, IDE pop up as shown in Fig 10.9.7. The red colour circle icon is used to record the test case. By default, it is in recording mode; we can pause and stop it.

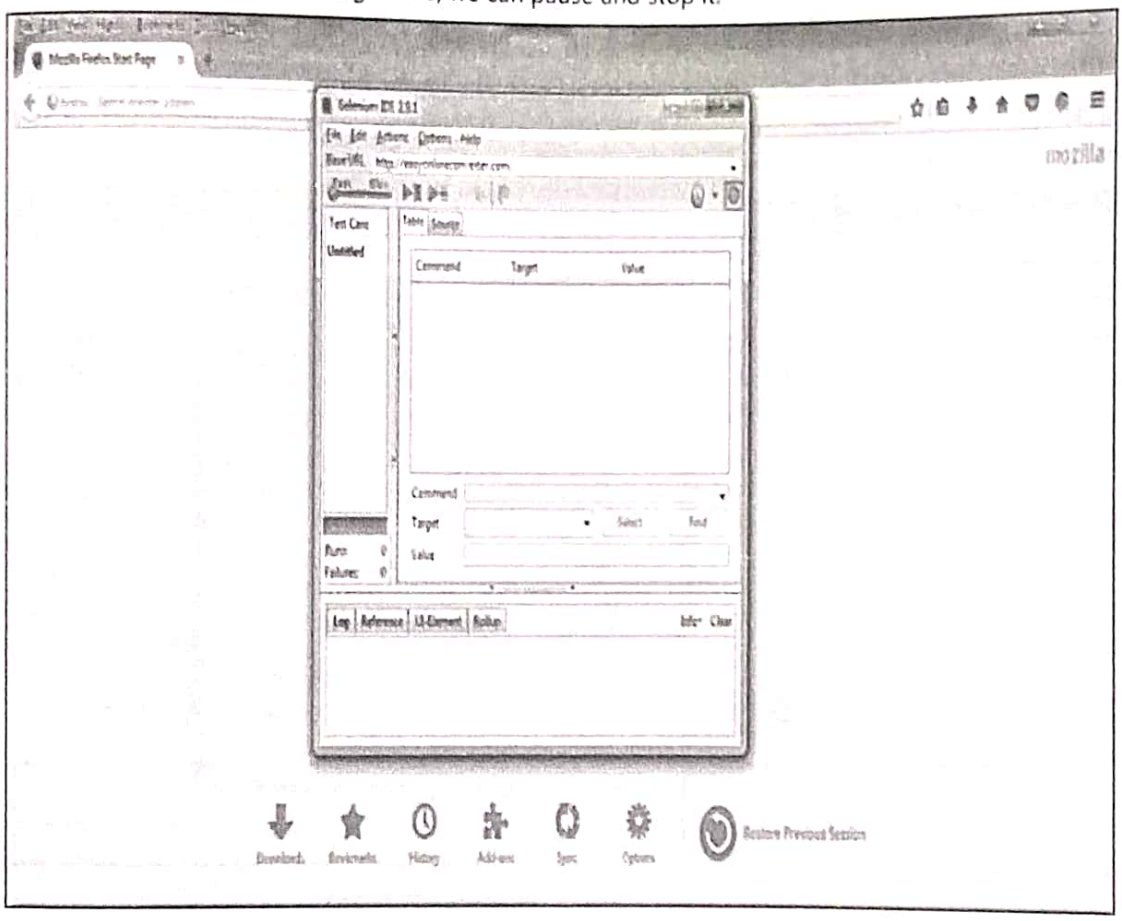

Fig. 10.9.7 : Selenium IDE pops up in FireFox.

Step 3 : Generating Selenium IDE Script

- Application to be tested: Open a new tab and enter the address of application under test (e.g.www.easyonlineconverter.com [10]), open text converter, and type some text in the text box, then change it to uppercase, lowercase, alternate case, sentence case etc.
- Recording: All this processing will be recorded into Selenium IDE as shown in Fig. 10.9.8. Selenium IDE supports the tester to record user interactions with the browser and therefore the complete recorded schedule is designated as Selenium IDE script.

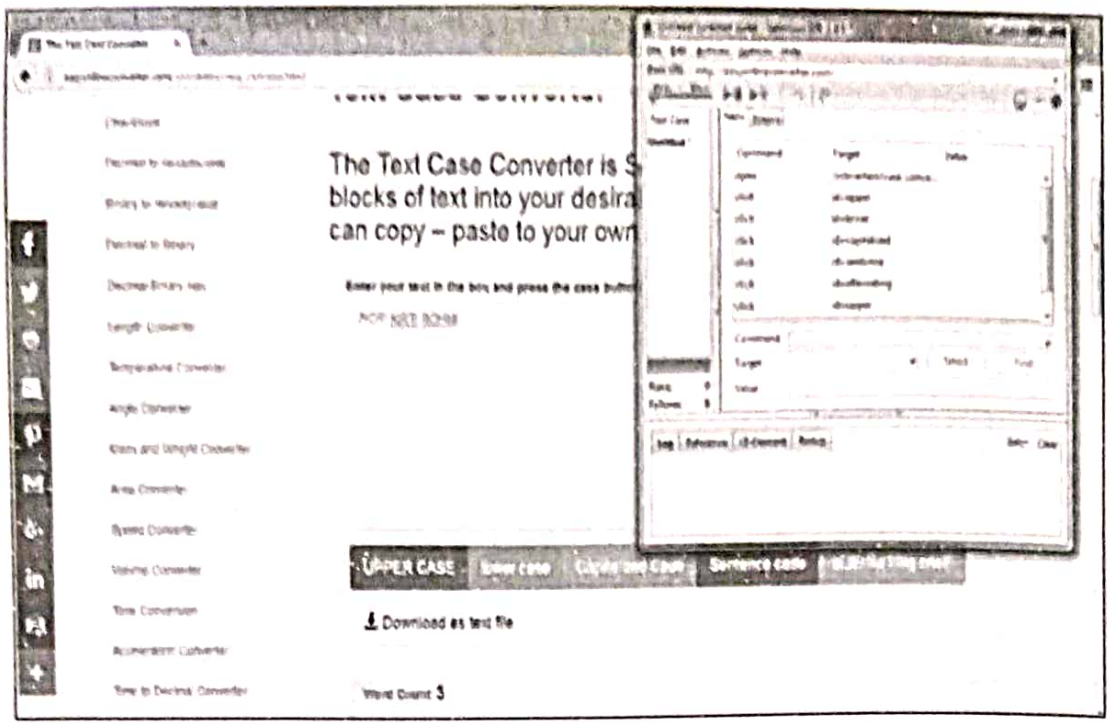

Fig. 10.9.8 : Recording of the Selenium IDE Script

- Save and Run Process- Once a script has been created, it can be saved for future runs as shown in Fig. 10.9.9. There is no failure in the current test run. We can slow the process of running by control button of fast and slow as shown in Fig. 10.9.9.

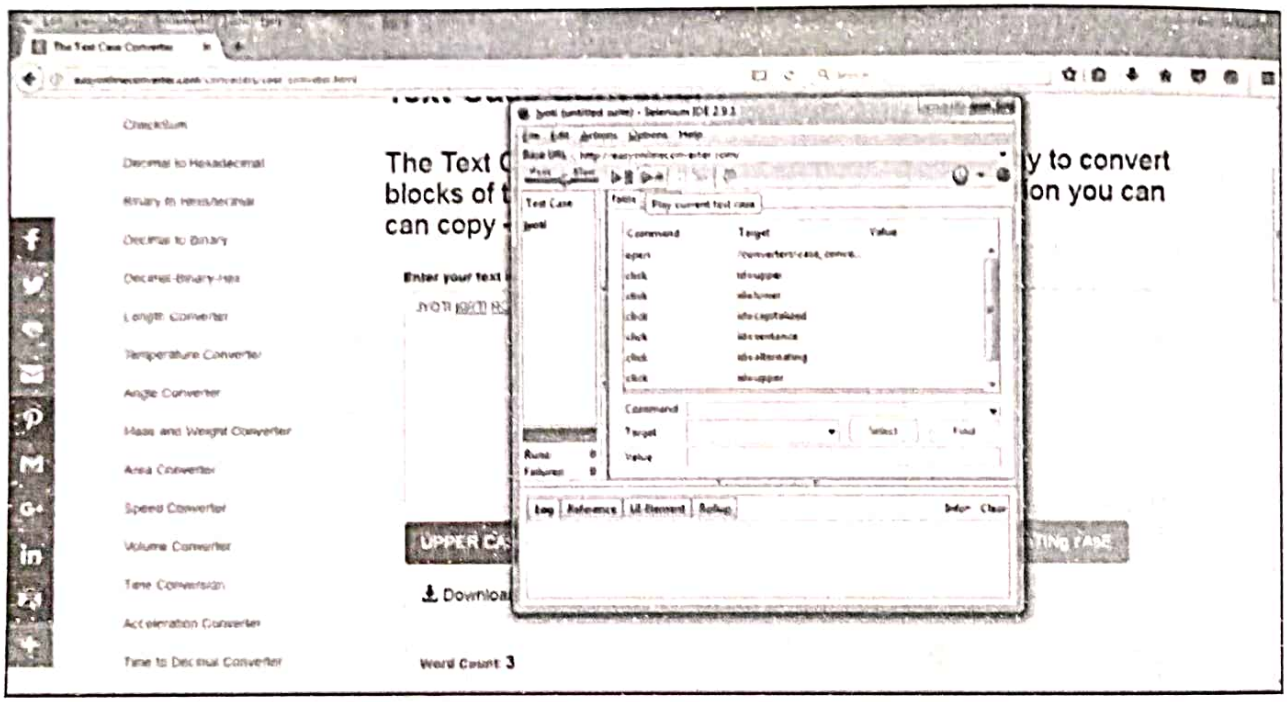

Fig. 10.9.9 : Run the Selenium IDE Script

10.9.5 Examination of Result

- After test run, its result can be checked in the log field as shown in Fig. 10.9.10. It shows that Test case has been passed and there is no failure. When the test case is executed, error and information messages which display the development in the log pane repeatedly, though the log tab has not been selected. These messages are frequently valuable for test case debugging.

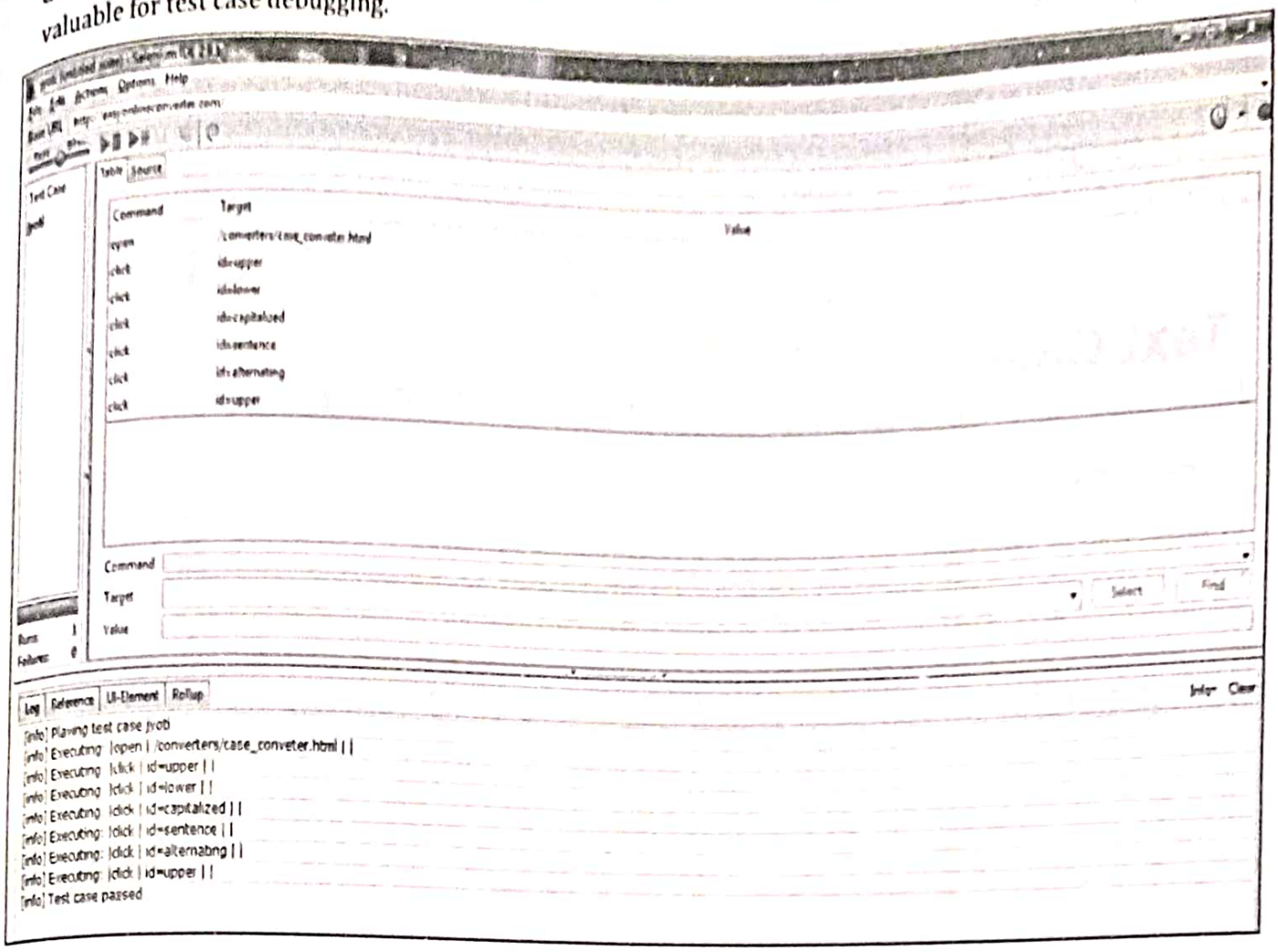

Fig. 10.9.10 : Result in log block

- The Clear button is used for clearing the Log and the Info button is a drop-down which allow the selection of diverse levels of information to log. The Reference tab is the default selection whenever the Selenium commands and parameters are entered or modified in Table mode. In Table mode, it will show documentation on the current command. When entering or modifying commands, whether from Table or Source mode, it is very significant to confirm that the parameters indicated in the Target and Value fields should be equal to those identified in the parameter list in the Reference pane. The Reference tab has been shown in the Fig. 10.9.11.

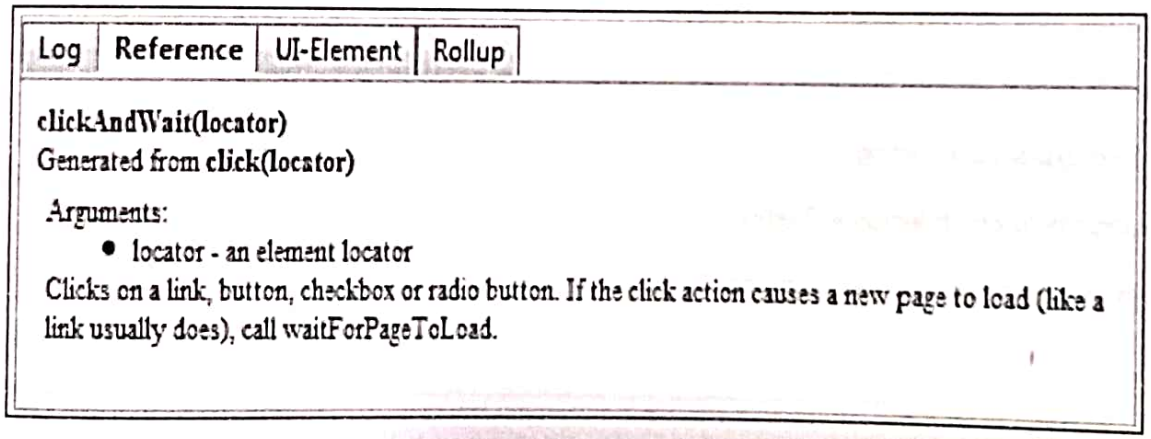

Fig. 10.9.11 : The Reference Tab

10.9.6 Exporting the Test Results

- The test case can be exported in either a Web Drive or Remote-Control Unit for future use, the language used for test case can be C#, JAVA, Python or Ruby, as shown in Fig. 10.9.12.

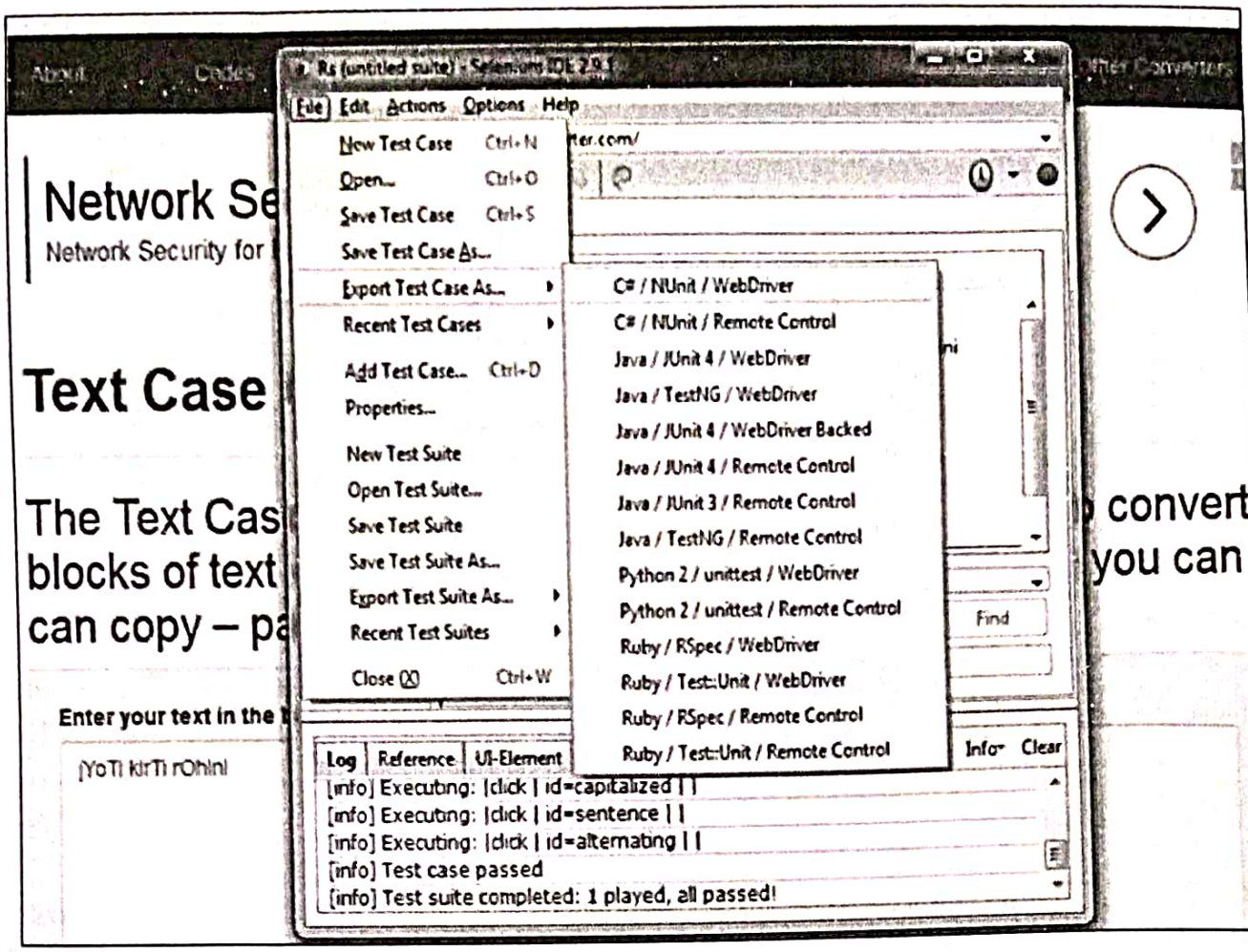

Fig. 10.9.12 : Export Test Case

- The selenium tool runs successfully one component of the application with no failure. In the setup phase, address of the base URL has been given, i.e. the address of the application under test.

Review Questions

1. Write short note on : Verification and Validation.
2. Explain Software Testing Strategy.
3. Draw and explain unit testing.
4. Write short note on : Integration Testing.
5. Explain test strategies for object-oriented Software.

□□□