SOFTWARE TEST AUTOMATION

SOFTWARE TEST AUTOMATION

DEVELOPING SOFTWARE TO TEST THE SOFTWARE IS CALLED TEST AUTOMATION.

ADVANTAGE OF TEST AUTOMATION

- **Automation saves time as software can execute test cases <u>faster</u> than human do.**
- **Test automation can free the test engineers from <u>mundane tasks</u> and make them focus on more creative tasks.**
- **Automated tests can be more <u>reliable</u>.**
- **Automation helps in <u>immediate</u> testing.**
- **Automation can <u>protect</u> an organization against attrition of test engineers.**
- **Test automation <u>opens up opportunities</u> for better utilization of global resources**
- **Certain types of testing cannot be executed <u>without automation</u> .**
- **Automation means <u>end-to-end</u>, not test execution alone.**

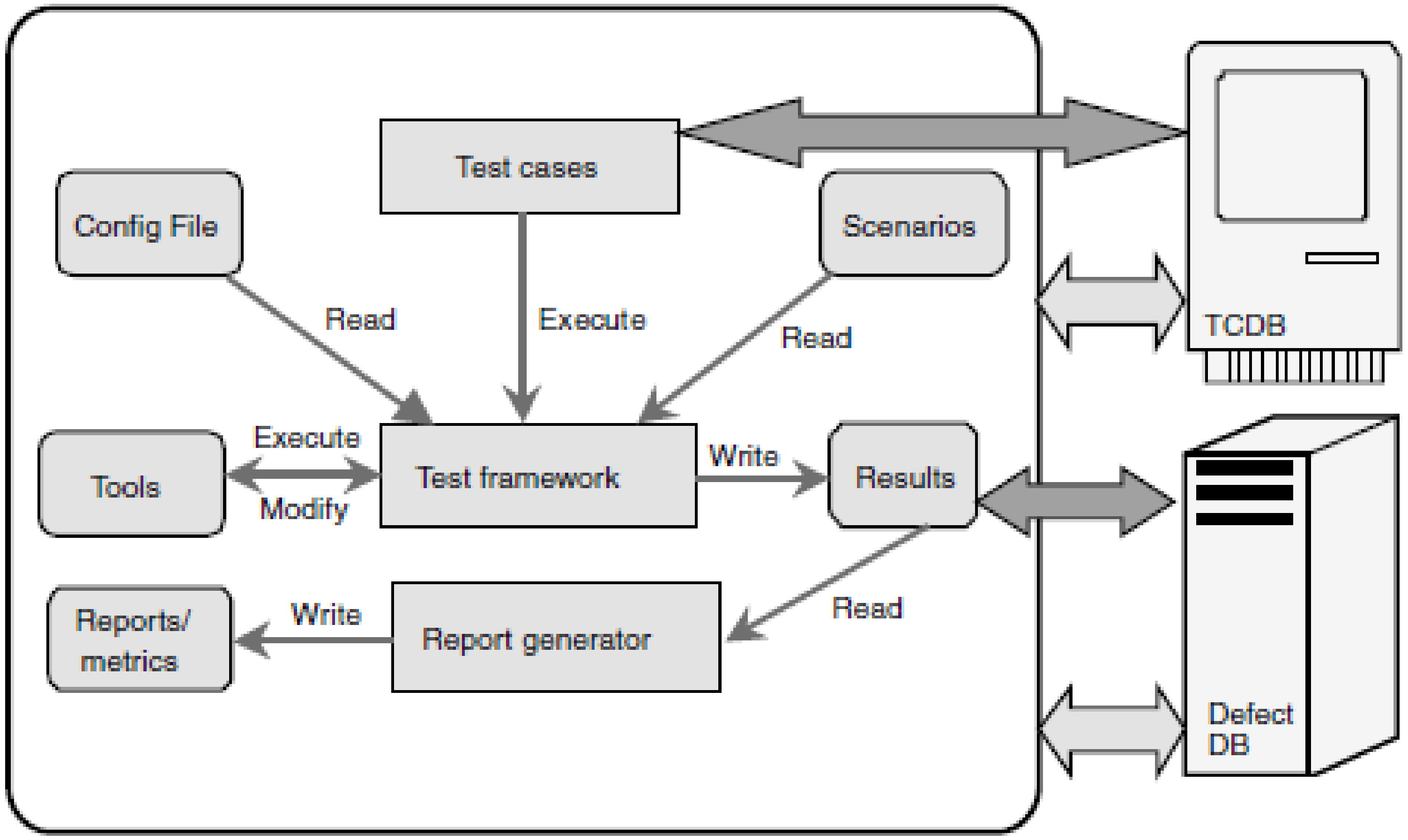
TEST AUTOMATION IN

S.No.	Test cases for testing	Belongs to what type of testing
1	Check whether log in works	**Functionality**
2	Repeat log in operation in a loop for 48 hours	**Reliability**
3	Perform log in from 10000 clients	**Load/stress testing**
4	Measure time taken for log in operations in different conditions	**Performance**
5	Run log in operation from a machine running Japanese language	**Internationalization**

DESIGN & ARCHITECTURE FOR AUTOMATION

- Design and architecture is an important aspect of automation.
- As in product development, the design has to represent all requirements in modules and in the interactions between modules.
- The test framework provides a backbone that ties the selection and execution of test cases. Components of test automation:
  - external MODULES
  - scenario AND CONFIGURATION FILE MODULES
  - test CASES AND TEST FRAMEWORK MODULES
  - tools AND RESULTS MODULES
  - report GENERATOR AND REPORTS/METRICS MODULES

DESIGN & ARCHITECTURE FOR AUTOMATION



DESIGN & ARCHITECTURE FOR AUTOMATION

External modules

- There are two modules that are external modules to automation—tcdb and defect db.
- All the test cases, the steps to execute them, and the history of their execution (such as when a particular test case was run and whether it passed/failed) are stored in the tcdb.
- Defect db or *defect database* or *defect repository* contains details of all the defects that are found in various products that are tested in a particular organization. It contains defects and all the related information (when the defect was found, to whom it is assigned, what is the current status, the type of defect, its impact, and so on).

DESIGN & ARCHITECTURE FOR AUTOMATION

Scenario and configuration file modules

- *Scenarios* are information on "**how to execute a particular test case**."

- A *configuration file* contains a **set of variables** that are used in automation. The variables could be for the test framework or for other modules in automation such as tools and metrics or for the test suite or for a set of test cases or for a particular test case.

DESIGN & ARCHITECTURE FOR AUTOMATION

Test cases and test framework modules

- Test case is an **object for execution** for other modules in the architecture and does not represent any interaction by itself.

- A *test framework* is a module that combines "**what to execute" and "how they have to be executed.**"

- It picks up the specific test cases that are automated from tcdb and picks up the Scenarios and executes them.
  - For example, if there is a scenario that requests a particular test case be executed for 48 hours in a loop, then the test framework executes those test cases in the loop and times out when the duration is met.

DESIGN & ARCHITECTURE FOR AUTOMATION

Tools and results modules

- When a test framework performs its operations, there are a set of tools that may be required.
    - For example, when test cases are stored as source code files in tcdb, they need to be extracted and compiled by build tools. In order to run the compiled code, certain runtime tools and utilities may be required.

- When a test framework executes a set of test cases with a set of scenarios for the different values provided by the configuration file, the results for each of the test case along with scenarios and variable values have to be stored for future analysis and action.

DESIGN & ARCHITECTURE FOR AUTOMATION

Report generator and reports/metrics modules

- Once the results of a test run are available, the next step is to prepare the test reports and metrics. Preparing reports is a complex and time-consuming effort and hence it should be part of the automation design.
- The module that takes the necessary inputs and prepares a **formatted report** is Called a *report generator.*
- Once the results are available, the report generator can generate *metrics*.
- All the reports and metrics that are generated are stored in the Reports/metrics module of automation for future use and analysis.

GENERIC REQUIREMENTS FOR TEST TOOL/FRAMEWORK

1. No hard coding in the test suite
2. Reuse of code for different types of testing, test cases
3. Test case/suite expandability
4. Automatic setup and cleanup
5. Independent test cases
6. Test case dependency
7. Insulating test cases during execution
8. Coding standards and directory structure
9. Selective execution of test cases
10. Random execution of test cases

GENERIC REQUIREMENTS FOR TEST TOOL/FRAMEWORK

8. Parallel execution of test cases
9. Looping the test cases
10. Grouping of test scenarios
11. Test case execution based on previous results
15. Remote execution of test cases
16. Automatic archival of test data
17. Reporting scheme
18. Independent of languages
19. Portability to different platforms

NO HARD CODING IN THE TEST SUITE

modular tc, adding one tc should not require full retest
adding new tc should not affect old tc's
thus flexibility

- Adding a test case should not affect other test cases
- Adding a test case should not result in retesting the complete test suite
- Adding a new test suite to the framework should not affect existing test Suites

R-1

TEST CASE/SUITE EXPANDABILITY

ability to be expanded to include new tc's
and not affect existing ones

- The test suite should only do what a test is expected to do. The test framework needs to take care of "how,"

- The test programs need to be modular to encourage reuse of code.

R-2

REUSE OF CODE FOR DIFFERENT TYPES OF TESTING, TEST CASES

framework should support reusablity at
functional, performance, regression testing

- For each test case there could be some prerequisite to be met before they are run.
- When test cases expect a particular setup to run the tests, it will be very difficult to remember each one of them and do the setup accordingly in the manual method.
- Hence, each test program should have a “setup” program that will create

The necessary setup before executing the test cases.

R-3

AUTOMATIC SETUP AND CLEANUP

- A setup for one test case may work negatively for another test case.
- Hence, it is important not only to create the setup but also “undo” the setup soon after the test execution for the test case.
- Hence, a “cleanup” program becomes important and the test framework should have facilities to invoke this program after test execution for a test case is over.

R-4

INDEPENDENT TEST CASES

- Making test cases independent enables any one case to be selected At random and executed.

- Making a test case dependent on an other makes it necessary for a Particular test case to be executed before or after a dependent Test case is selected for execution.

R-5

TEST CASE DEPENDENCY

- Insulating test cases from the environment is an important requirement for The framework or test tool.

- At the time of test case execution, there could be some events or interrupts or signals in the system that may affect the execution.

- Consider the example of automatic pop-up screens on web browsers. When such pop-up screens happen during execution, they affect test case execution as the test suite may be expecting some other screen based on an earlier step in the test case.

R-6

INSULATING TEST CASES DURING EXECUTION

insulated from external interruptions like
pop ups or system events

- To avoid test cases failing due to some unforeseen events, the framework Should provide an option for users to block some of the events.

- There has to be an option in the framework to specify what events can affect The test suite and what should not.

R-7

CODING STANDARDS AND DIRECTORY STRUCTURE

enforce coding style and directory structure for consistency

- A framework may have multiple test suites; a test suite may have multiple test programs; and a test program may have multiple test cases.
- The test tool or a framework should have a facility for the test engineer to select a particular test case or a set of test cases and execute them.
- The selection of test cases need not be in any order and any combination should be allowed.

R-8

SELECTIVE EXECUTION OF TEST CASES

to provide granular control over test execution

Example:

- Test-program-name 2, 4, 1, 7-10
- In the above scenario line, the test cases 2, 4, 1, 7, 8, 9, 10 are selected for execution in the same order mentioned.

R-9

RANDOM EXECUTION OF TEST CASES

to uncover defects caused by unexpected test order

- Giving a set of test cases and expecting the test tool to select the test case is called random execution of test cases.
- A test engineer selects a set of test cases from a test suite; selecting a random test case from the given list is done by the test tool.

Example 1:	**Example 2:**
random test-program-name 2, 1, 5	random test-programl (2, 1, 5) test-program2 test-program3

R-10

PARALLEL EXECUTION OF TEST CASES

- In a multi-tasking and multi processing operating systems it is possible to make several instances of the tests and make them run in parallel.
- Parallel execution simulates the behavior of several machines running the same test and hence is very useful for performance and load testing.

Example 1:	**Example 2:**
instances,5	Time_loop, 5 hours
Test-program1(3)	test-program1(2, 1, 5)
	test-program2
	test-program3

R-11

LOOPING THE TEST CASES

R-12

- Reliability testing requires the test cases to be executed in a loop.
- There are two types of loops that are available.
- One is the *iteration* loop which gives the number of iterations of a particular test case to be executed.
- The other is the *timed* loop, which keeps executing the test cases in a loop till the specified time duration is reached.

Example 1:	**Example 2:**
Repeat_loop, 50 Test-program1(3)	Time_loop, 5 hours test-program1(2, 1, 5) test-program2 test-program3

GROUPING OF TEST SCENARIOS

to facilitate complex test strategies

- We have seen many requirements and scenarios for test execution.
- The group scenarios allow the selected test cases to be executed in order, random, in a loop all at the same time.
- The grouping of scenarios allows several tests to be executed in a predetermined combination of scenarios.

Example:

```
group_scenario1
        parallel, 2 AND repeat, 10 @ scen1
scen1
        test_program1 (2, 1, 5)
        test_program2
        test_program3
```

conditional execution

TEST CASE ~~EXECUTION~~ BASED ON PREVIOUS RESULTS

- Some of the common scenarios that require test cases to be executed based on The earlier results are
  1. Rerun all test cases which were executed previously;
  2. Resume the test cases from where they were stopped the previous time;
  3. Rerun only failed/not run test cases; and
  4. Execute all test cases that were executed on "sept 26, 2020."

- With automation, this task becomes very easy if the test tool or the framework can help make such choices.

R-14

REMOTE EXECUTION OF TEST CASES

execute, stop and monitor tests
from central console in distributed machines

- It should be possible to execute/stop the test suite on any machine/set of Machines from the test console.
- The test results and logs can be collected from the test console.
- The progress of testing can be found from the test console.

R-15

AUTOMATIC ARCHIVAL OF TEST DATA

- Every test suite needs to have a reporting scheme from where meaningful reports can be extracted.
- Though the report generator is designed to develop dynamic reports, it is very Difficult to say what information is needed and what not.
- Therefore, it is necessary to store all information related to test cases in the results file.

R-16

R-17

REPORTING SCHEME

acutal contents in previous slide

- A framework should be independent of programming languages and scripts.
- A framework should provide choice of programming languages, scripts, and their combinations.
- A framework or test suite should not force a language/script.
- A framework or test suite should work with different test programs written using different languages and scripts.
- A framework should have exported interfaces to support all popular, standard languages, and scripts.
- The internal scripts and options used by the framework should allow the Developers of a test suite to migrate to better framework.

INDEPENDENT OF LANGUAGES

acutal contents in previous slide

- With the advent of platform-independent languages and technologies, there are many products in the market that are supported in multiple os and language platforms.
- Products being cross-platform and test framework not working on some of those platforms are not good for automation.
- Hence, it is important for the test tools and framework to be cross-platform and be able to run on the same diversity of platforms and environments under which the product under test runs.

R-18

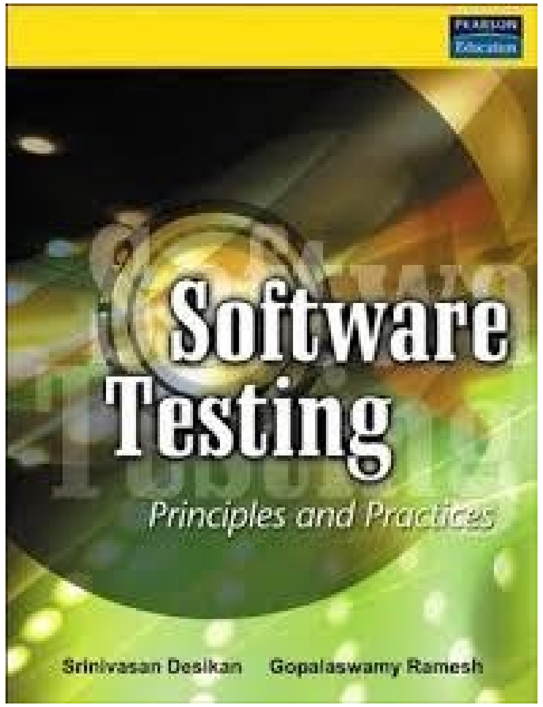
PORTABILITY TO DIFFERENT PLATFORMS

acutal contents in previous slide and this slide

- The framework and its interfaces should be supported on various platforms.
- Portability to different platforms is a basic requirement for test tool/test Suite.
- The language/script used in the test suite should be selected carefully so that it run on different platforms.
- The language/script written for the test suite should not contain platform- specific calls.

R-19

REFERENCES



Guru99

edureka!

Software Testing Help