Chapter - 2

Survey of Literature
CHAPTER – 2

SURVEY OF LITERATURE

2.1 AUTOMATION TESTING FRAMEWORKS

The chapter will describe the key benefits of software test automation and examine implementation. It will then analyze some of the key reasons why test automation has not met with the level of success that it could. It also introduces about Component Testing and Accounting Center. It will examine how using a frame worked approach to test automation can allow organizations to avoid the problems inherent in other approaches, and realize the benefits of test automation.
The Software testing is an integral, costly and time-consuming activity in the software development life cycle. Since testing involves running the system under test under variety of configuration and circumstances, automation of testing activities is a potential source of saving in testing process. Testing a complex system (like UMTS, GPRS, SoftSwitch) calls for sophisticated test tools that are required at almost every phase of the testing.

Calls for sophisticated test tools that are required at almost every phase of the testing development of such tools requires time & effort. This calls for a need to have a generic test framework, which can be adapted to perform in diverse scenarios, test the item to the Maximum possible extent and provide performance and Capability comparable to the standard test tools. Requirement of a generic test framework becomes more crucial if the interfaces/protocols are Non-standards.

2.2 THE BENEFITS OF SOFTWARE TEST AUTOMATION

Most, software development and testing organizations are well aware of the benefits of test automation. A Quick glance at the web sites of any test automation tool vendor will point out a number of the key benefits of test automation. Some of these benefits include:

A. Reduce test executing time and cost Automated tests take less time to execute than manual tests, and can generally execute unattended. A tester must simply start the test, and then analyze the results when the test is completed.
B. Increased test coverage on each testing cycle. Automated test can allow testing teams to execute large volumes of tests against each build of their application, achieving a level of coverage that would not be possible with manual testing. This increased coverage can help teams uncover bugs in existing functionality much more quickly than through manual testing. Test automation can allow teams to test more features in each cycle (breadth), and also to test features using more permutations of inputs (depth).

C. Increase value of manual testing effort. So long as applications are meant for human end users, test automation will never entirely replace the need for human testers. No matter how sophisticated test automation tools become, they will never be as good as human testers at finding bugs in an application. Human testers will instantly notice subtle bugs that are almost never detected by test automation, particularly usability bugs. Automated test tools cannot 'follow their instincts' to uncover bugs using exploratory and ad hoc testing techniques. By freeing manual testers from having to execute repetitive, mundane tests, test automation enables them to focus on using their creativity, knowledge, and instincts to discover important bugs.

2.3 WHY TEST AUTOMATION PROJECTS BE UNSUCCESSFUL TO PULL OF THEIR POTENTIAL

Despite the clear benefits of test automation, many organizations are not able to build effective test automation programs. Automating testing process is a
challenging task for any organization if it is not done in a proper way than test automation becomes a costly effort that finds few bugs and is of questionable value to organization. There are a number of reasons why test automation efforts are unproductive. Some of the most common include:

A. Poor quality of tests being automated

Mark fewster explains this problem very well “It doesn’t matter how clever you are at automating a test or how well you do it, if the test itself achieves nothing then all you end up with is a test that achieves nothing faster “[fewster, software Test Automation, I.1, (Addison Wesley) 1999]"

Many organizations simply focus on taking existing rest cases and converting them into automated tests. There is a sense that if 100% of the manual test cases can be automated, then the test automation effort will be a success. In trying to achieve this goal, many organizations find that they may have automated many of their manual tests, but it has come at a huge investment of time and money, and produces few bugs found. This can be due to the fact that a poor test is a test, whether it is executed manually or automatically.

B. Lack of good test automation tools and process

Many teams acquire a test automation tool and begin automating as many test cases as possible, with little consideration of how they can structure their automation in such a way that it is scalable and maintainable. Little consideration is given to
managing the test scripts and test results, creating reusable functions, separating data from tests and other key issues, which allow a test automation effort to grow successfully. After sometime, the team realizes that they have hundreds or thousands of test scripts, thousands of separate test result files, and the combined work of maintaining the existing scripts while continuing to automate new ones requires a larger and larger test automation team with higher and higher costs and no additional benefit.

Inability to adapt to changes in the system under test as teams drive towards their goal of automating as many existing test cases as possible, they often don’t consider what will happen to the automated tests when the application under test (AUT) under goes a significant change.

Lacking a well conceived test automation framework that considers how to handle changes to the system under test, these teams often find that majority of their test scripts need maintenance the Outdated scripts will usually result in skyrocketing numbers of false negatives, since the scripts are no longer finding the behavior they are programmed to expect. As the team hurriedly works to update the test scripts to account for the changes, project stakeholders begin to lose faith in the results of the test automation. Often times the lack of perceived value in the test automation will result in a decision to scrap the existing test automation effort and start from scratch, using a more intelligent approach that will produce incrementally better results.
2.4 GENERATIONS: TEST AUTOMATION EVOLUTION

Software test automation has evolved through several generations of tools and techniques:

Capture/playback tools record the strokes of tester in a manual test, and allow tests to be run unattended for many hours each day, greatly increasing test productivity and eliminating the mind-numbing repetition of manual testing. However, even small changes to the software under test require that the test be recorded manually again. Therefore this first generation of tools is not efficient or scalable.

Scripting, a form of programming in computer languages specifically developed for software test automation, alleviates many issues with capture/play back tools. However, the developers of these scripts must be highly technical and specialized programmers who work in isolation from the testers actually performing the tests. In addition, scripts are best suited for GUI Testing and don’t lend themselves to embedded, batch, or other forms of systems. Finally, as changes to the software under test require complex changes to the associated automation scripts, maintenance of ever-larger libraries of automation scripts becomes an overwhelming challenge.

Data-driven testing is often considered separately as an important development in test automation. This approach simply but powerfully separates the automation script from the data to be input and expected back from the software under
test. This allows the data to be prepared by testers without relying on automation engineers, and vastly increases the possible variations and amounts of data that can be used in software testing. This breaking down of the problem into two pieces is very powerful. While this approach greatly extends the usefulness of scripted test automation, the huge maintenance chores required of the automation programming staff remain.

Keywords-based test automation breaks work down ever further, in an advanced, structured and elegant approach. This reduces the cost and time of test design, automation, and execution by allowing all members of a testing team to focus on what they do best. Using this method, non-technical testers and business analysts can develop executable test automation using “keywords” that represent Strokes recognizable to end-users, such as “login”. While automation engineers devote their energy to coding the low level steps that make up those strokes, such as “click”, “find text box A in window B” “enter User Name”, etc. Keyword-based test design can actually begin based on documents developed by business analysts or the marketing department, before the final details of the AUT are known. As the test automation process proceeds, bottlenecks are removed and the expensive time of highly trained professionals is used effectively.

The cost-benefits of the keyword method become even more apparent as the testing process continues. When the software under test undergoes changes, revisions to the test and to the automation scripts are necessary. Organizing test design and test automation with the keyword framework eliminates time previously allocated to maintaining large libraries of scripts and rewriting entire scripts a new after major
changes to the software under test. With the keyword method, the necessary changes are far fewer. Many changes do not require new automation at all, and can be completed by non-technical testers or business analysts. When required, changes to automated keywords can be completed by automation engineers without affecting the rest of the test.

2.5 TEST AUTOMATION FRAMEWORK SELECTION

Basing an automated testing effort on using only a capture tool such as IBM Rational Robot or Mercury Win Runner to record and play back test cases has its drawbacks. Running complex and powerful tests is time consuming and expensive when using only a capture tool. Because these tests are created ad hoc, their functionality can be difficult to track and reproduce, and they can be costly to maintain.

A better choice for an automated testing team that’s just getting started might be to use a test automation framework, defined as a set of assumptions, concepts, and practices that constitute a work platform or support for automated testing. Some driving parameters for automation framework are:

The following are the driving forces that lead to framework.

- Accessibility

This is the quality of acquiring any resource when required at any point of time.

- Open Standards
This will bring in full of advantages of open standards like xml.

- **Platform Independence**

  The test suites are implemented on remote platforms. This gives the programmer an edge to combine open ended system for proprietary platforms.

- **Usability**

  Existing tools that come along with OS can be used. No more extra costs involved.

- **Security**

  The high level of security will be maintained.

- **Changes**

  The existing test suites need not be rewritten and can be modified in the least possible manner to comply with this frame.
A. The Test Script Modularity Framework

The test script modularity framework requires the creation of small, independent scripts that represent modules, sections, and functions of the application-under-test. These small scripts are then used in a hierarchical fashion to construct larger tests, realizing a particular test case.

B. The Test library Architecture Framework

The test library architecture framework is very similar to the test script modularity framework and offers the same advantages, but it divides the application-under-test into procedures and functions instead of scripts. This framework requires the creation of library files (SQABasic libraries, APIs, DLLs and Such) that represent modules, sections, and functions of the application-under-test. These library files are then called directly from the test case script.
C. The Keyword-Driven or Table-Driven Testing Framework

Keyword-driven testing and table-driven testing are interchangeable terms that refer to an application-independent automation framework, independent of the test automation tool used to execute them and the test script code that “drives” the application-under-test and the data. Keyword-driven tests look very similar to manual test cases. In a keyword-driven test, the functionality of the application-under-test is documented in a table as well as in step-by-step instructions for each test.

D. The Data-Driven Testing Framework

Data-driven testing is a framework where test input and output values are read from data files (data pools, ODBC sources, csv files, Excel files, DAO objects, ADO objects, and such) and are loaded into variables in captured or manually coded scripts. In this framework, variables are used for both input values and output verification values. Navigation through the program, reading of the data files, and logging of test status and information are all coded in the test script. This is similar to table-driven testing in that the test case is contained in the data file and not in the Script; the script is just a “driver”, or delivery mechanism for the data. Unlike in table-driven testing, though, the navigation data isn’t contained in the table structure. In data-driven testing, only test data is contained in the data files.
E. The Hybrid Test Automation Framework

The most commonly implemented framework is combination of all of the above techniques, pulling from their strengths and trying to mitigate their weaknesses. This hybrid test automation framework is what most frameworks evolve into over time and multiple projects.

2.6 HYBRID TESTING FRAMEWORK

We have discussed different types of frameworks. Every framework have there own benefits and drawbacks standalone no one is perfect for automation testing with concern of performance and reusability constraints. The most commonly implemented framework is a combination of all of the above techniques, pulling from their strengths and trying to mitigate their weaknesses. This hybrid test automation framework is what most frameworks evolve into over time and multiple projects. Figure 2.2 gives you an idea of how you could combine the approaches of the different frameworks within mercury Win Runner. The TAF. Approach, as illustrated in the figure above, reflects how test engineers develop the logical variations of information associated with data and control for the required functionality of an application using models.
Traditional test design begins The Stroke Based Testing Framework with a written narrative that must be interpreted by each tester or automation engineer working on the test.

These models are then translated and tests are generated of the logical combination. The test engineer must develop (only once) a test driver schema that is a pattern for all test drivers.

Some key advantages of this approach include:

1. All Models can use the same test driver schema to produce test scripts for the requirements captured in each model
2. When system functionally changes or evolves, the logic in the models change. and all related test are regenerated using the existing test driver schema.
3. If the test environment changes, only the test driver schema needs modification. The test drivers associated for each model can be regenerated without any changes to the model
Following are the individual component explanation of hybrid framework:
Fig 2.2 Hybrid test automation testing

Application Table- a repository of all applications to be tested

Application Driver – Reads selected application and calls the Startup script.

Startup Script – invokes the application, checks the system configuration and available memory, and calls the `core table driven engine`.

Cycle Table – high level table listing suites of tests to execute

Cycle Driver – reads each record from the Cycle Table and passes the selected test suites to the Suite Driver.

Library – contains common test case procedures.

Suite Driver – reads each record from the Suite Table and passes them to the Step Driver.

Step Table – records of low-level instructions developed in the keyword vocabulary of the functions.
Step Driver – parses the Step Table and performs the initial error detection, correction, and Synchronization.

Functions – application independent functions that actively manipulate component objects.

Applications Functions – modules that are specific to the application under test

Monitor – synchronizes test and checks for errors and exceptions

Test log – reports all test activities

Error log – logs all unexpected errors

Error Handler – verifies the errors and exceptions and checks if application can recover

Recovery – configures the application to its original state

Recovery log – contains all the error recovery actions

Shut Down – closes the application under test

Clean Up – deletes any unnecessary files created during the test.
2.7 COMPONENT TESTING

Component testing includes the following activities for any given test case according to test standard.

(1) Component testing program: It stipulates how test strategies and component test plan is used in components which are to be tested and the software which will communicate with testing components, such as drive and pile.

(2) Specifications of component testing: test cases are designed by test cases designing technology, which has been chosen in test program activities. In the use of the test case design technology choices to. Test specification should include test objective, initial state of components, input and expected output. Implementation of each test case is repeatable.

(3) Implementation of component test: Each test case should be enforceable.

(4) Record of component test: Explicit signification and version record of test specification and components, which are to be tested, should be included in each record of test case. Actual output should also be put on record, for all specific test events will be established by these test records. In addition, actual and expected output should be compared, any difference should be recorded and analyzed to find mistakes. and early test activities should be established to be the basis of deleting specification differences and defects of components.
(5) Examine whether component test is finished: Check whether test record is compatible with previous defined test standards. If these standards are not met, early test activities must be repeated, and then begin testing process from this point. It may also need to repeat test specification activities, and further design test case to meet the requirement of test coverage objectives.

2.8 COMPONENT TEST STRATEGY WITH OVERALL PROJECT STRATEGY

There is a need for all project activities to contribute to the attainment of the project's long term goals. In particular, both the component-level and system-level testing strategies should be compatible with the project's development strategy.

The component testing process is often neglected in the sense that it is not formally planned. It is assumed that developers will not submit to the code base anything that they have not tested; however, different developers have different views on what constitutes adequate testing. If left unspecified, this effort produces varying levels of quality.

Many companies have a testing group that is independent of the development groups. Testing is viewed as "their" responsibility. The developers are assumed to do some level of testing on the individual components that they construct but no formal plan is created to guide this activity. The lack of communication between the development group and the testing group results in insufficient attention to testing on
the development side and insufficient knowledge of the development process on the testing side. An increasing number of companies face external forces, such as regulatory authorities, that impose requirements on the testing strategy. These requirements include producing evidence that the agency's standards have been addressed. A component test plan is integral to this evidence. Information required for the component testing process may actually improve the development process. Pictorial models, such as the object/dynamic/functional models, are essential for good test planning and are valuable for the development effort as well. Having testing and development organizations communicate during the project planning phase can result in an improved development process as well as improved quality.

Include an explicit plan for component testing as an integral part of the project's test plan. Assure that the component test plan complements the system test plan and is compatible with the overall goals of the development plan. Do this by enumerating the development goals of the project and identifying compatible goals for the testing process.

- Identify and prioritize the development goals for the project.
- Determine those that impact the component level of the system.
- For each selected development goal, establish a matching, and complementary component testing goal.
2.9 HISTORY OF ACCOUNTING CENTER

The Accounting Center is the future state gateway for all financial transactions feeding into the General Ledger and Merlin. Accounting Center will also consist of an integration hub layer facilitating data transfer to and from the integrated future state PeopleSoft system consisting of Accounts Payables, Treasury, Asset Management and Travel and Expense modules.

The Accounting Center will also be a repository for detail transactions to be stored for reporting and querying purposes. The Accounting Center is a major enabler of the composite solution for the Finance Home Run Program. The implementation of the complete solution of the Accounting Center will be accomplished via four work streams

1. **Accounting Center POC**: POC vendor products (selected solution alternative in Concept Phase) new to CNA’s environment. Ensure that Oracle Financial Services Accounting Hub (FSAH) and PM4data meet functional, non-functional, and technical requirements. Working with Oracle, incorporate key missing functionality within FSAH in future release of the product. Determine a buy vs. build approach based on the outcome of the POC.

2. **Accounting Center Version 1**: Deploy alternatives selected in the POC (1) utilizing current state financial language for selected feeds and deploying selected functionality (to be defined during definition) of the Accounting
Center. Deliver a fully functional Integration Hub layer to facilitate data movement.

3. **Accounting Center Version 1.5:** Create historical Unearned Master file in Accounting Center, balancing to production Stat Management Unearned File, and reloading Merlin MIS_EARN_OUT historically from data in the Unearned Master File in Accounting Center.

4. **Accounting Center Capability Version 2:** Deploy a capability that will encompass most source system feeds into the Accounting Center for editing, transformation and processing source data. In addition provide the capability to reconcile with the production feeds. The system will contain, auditable and reconcilable detailed information providing linkage to the source systems. In addition Version 2.0 will deploy the Accounting Center Journal Processing Center (JPC). The JPC is the end user application that will be the mechanism for journal processing and related journal functions. In addition the JPC will be used for data rule entry and maintenance. The JPC will be an instance of PeopleSoft GL 9.0. Version 2.0 also includes full integration to PeopleSoft GL 8.9 and integration with the Merlin views.

**Accounting Center Capability Version 2.5:** Deploy any remaining source system feeds in the planned future state financial language; in addition deploy any remaining functionality to the Journal Processing Center. Version 2.5 will also focus on detailed parallel runs of all interfaces and reconciling with the Production feeds. Version 2.5 will also contain the application roll out to the entire user base.