AUTOMATED TESTING *and SPI*

Brian Lynch

Introduction

The following document explains the purpose and benefits for having an Automation Test Team, the strategy/approach taken by an Automation Test Team and the contingencies for successful implementation into SQM.

Automated Testing and Software Process Improvement

Automated Testing will improve the software development process. As with any other process improvement activity it cannot be accurately assessed in isolation. It is dependent on other activities in the process to provide the most benefit.
The quantity and quality of test cases developed will be affected by the quantity and quality of the Requirements and Design specifications they are based on. The efficiency of the automated scripts is affected by the quantity and quality of the test cases *they* are based on. The quantity and quality of automated regression test scripts free up time testers have to find defect.
The earlier defects are found the less costly the software development effort.
A common theme for all software process improvement efforts is "the earlier the better" For example, it is known that the cost of finding defects later in the life cycle the more costly. This theme also applies to automated scripting. Although this does drastically reduce the cost of manual regression testing the true benefit of developing automated test scripts cannot be assessed until we factor in the time spent in tool training, the time to develop scripts and frequency of their execution and flexibility of scripting code.

Cost vs Benefit

Prior to Selection		**During Implementation**	
Cost Considerations	**Benefit Considerations**	**Time to develop automated scripts and frequency of their execution.**	
Purchase/price of tool Training and Support Setup of tool Maintenance of scripts Hiring of skilled/experienced scripters/programmers	Fewer hours/days/weeks needed Faster lead time Higher quality (no human error when executing) Higher motivation/productivity of personnel More test depth (as tool covers regression, testers focus on finding defects)	The more complex the applications and the longer the life cycles	The more time to develop, The more added value as automated scripts executions increase
		The less complex the application and the shorter the life cycles	The less time to develop scripts The less added value as automated scripts executions are less

See Graph Below

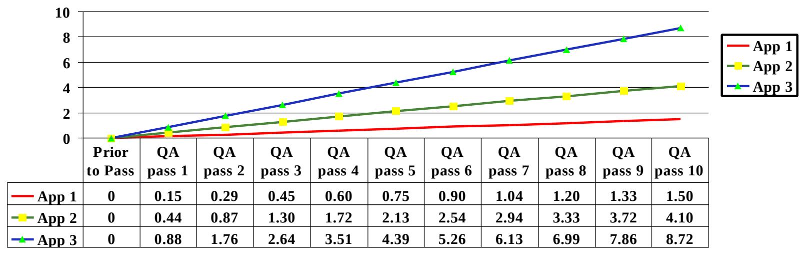
The earlier the automated scripts are developed the more they will be used and, therefore, the more beneficial they become. This can be illustrated in the formula below.

- Efficiency of Test Automation:

$$E = \frac{T1 * N}{T2 + T3 * N} = \text{Efficiency of Scripting}$$

T1 =	manual execution of Test Plan
T2 =	automated script effort
T3 =	automated execution of Test Plan
N =	number of QA passes (regressions)

- Efficiency of Scripting (E) vs Number of Regressions



	Prior to Pass	QA pass 1	QA pass 2	QA pass 3	QA pass 4	QA pass 5	QA pass 6	QA pass 7	QA pass 8	QA pass 9	QA pass 10
App 1	0	0.15	0.29	0.45	0.60	0.75	0.90	1.04	1.20	1.33	1.50
App 2	0	0.44	0.87	1.30	1.72	2.13	2.54	2.94	3.33	3.72	4.10
App 3	0	0.88	1.76	2.64	3.51	4.39	5.26	6.13	6.99	7.86	8.72

Role of Automation Test Team

The goal of the Automation Test Team (ATT) is to develop automated test scripts to facilitate various types of testing throughout the SDLC (software development life cycle). Testing and/or deliverables influencing/effecting testing occurs throughout the SDLC.

TEST Execution	***Functional/Regression***	*Performance/Scalability*
What	Business Functionality	Systems (Client, Network, Server) Performance
When in SDLC	Test Phases	Early and Throughout
How (Tool Used)	***Functional/Regression Tool***	***Load Test Tool***
Requirements	Test Cases	Usage Model and SLA
Application State	Stable with minimal UI Changes	Not Reliant on Front End

It is important that the automation team work closely *with* the QA, Development and Infrastructure teams to assess and prioritize which modules/functions/transactions to automate (see next page). The Development teams need to communicate any design-related issues (i.e. User Interface, Links, Page(s), changes in physical descriptions of objects and controls, etc.). The Infrastructure teams need to communicate any server architecture-related issues (i.e. hardware. firewalls, load balancing, server configurations, communication protocol, etc.).

Functional/Regression Test Scripting

Whereas most types of testing are intended to find defects, regression testing is not. It *may* find bugs but that is not the intention. Each defect that is found necessitates a code change. The regression test bed is constantly updated so that it reflects the current (i.e. test cases to accommodate code fixes) condition of the software. Once the change to the software is made, re-testing occurs and the cycle repeats.
The functional test-scripting tool is to facilitate regression **testing**.

Which Tests to Automate	**Which Tests Not to Automate**
Tests that need to be run for every build of the application (sanity level)	Usability testing – how easy is the application to use
Tests that use multiple data values for the same actions (data driven tests)	One time testing
Tests that require detailed information from application internals (e.g., SQL, GUI attributes)	Tests that need to be tested NOW
Stress / load testing	Ad hoc / random testing Based on intuitive knowledge of application
	Tests without predictable results
The more tedious the execution of the test case the more of a candidate it is for automation	The more creativity needed for test case execution, the less appropriate it is for automation.
	When cost – both product itself and/or cost of developing scripts outweigh benefit (see Efficiency of Test Automation above)

Automated scripts are based on regression test cases that have clear and expected results. Any test case that has clear and expected results implies that the program is functioning *as* expected. If the testing (manual or automated) finds actual results that do *not* match expected results implies there *is* some sort of unexpected result or defect whether it is static or dynamic. ***The scripts are only as effective as the test cases they are based on; if the test case does not elicit a defect, neither will the script.***

Strategy and Procedures

Automated Test scripting is a development effort and follows a similar process as far as iterative phases are concerned. What differs is the nature of the requirements, the Users and the final product. The goal is to automate as many *relevant* test cases within the approved Test Plans.
(Relevancy of test cases in regards to automation will have to be a joint decision between those writing the test cases and those designing the automated test script.)

Script Requirements

When developing an automated test strategy the first major activity is the identification of scripting requirements. For our purposes this will be derived from the test cases selected from the Test Plans.
They should contain the information to be entered (data), where it should be entered (field/window location), the navigational steps and the expected results. (Expected results, both functionality and performance, should be defined for both valid and invalid data - negative testing.)
It is at this point that the team:

- determine whether the test cases(s) is appropriate for automating

- Agree on which test cases are to be automated and their priority (i.e. those functions/modules/transactions having the highest amount of exception reports might indicate that that test case should be automated before a less "troubled" area of the application.)

Script Design

Once the scripting requirements have been agreed upon between QA and Automation team, the person designated to develop the automated test scripts will design the scripting for her/his particular test case. This will include such things as:

- development of AUT (application under test)-specific Initialization scripts
- construction of AUT-specific GUI file(s)
- determining how data will be inserted into script(i.e. hard coded vs. externally driven)
- if and where to loop
- where conditional statements should be included
- how to perform exception handling
- where and how to report results

There is a set of "best practices" to follow when developing automated regression test scripts.

1. Each TC has to be independent from other TCs. This means that it does not relay on the result of the previous TC. It has to drive the application from the initial stage of the app to the window where the checking of the TC's feature will take place After the verification is done it has to return the app in the initial stage.
   If a TC described in the Test plan relays on the result of another TC, a rule mentioned above has to apply to scripting of this TC.
2. Each TC has to check only one feature of the application. If in the Test plan a TC says that more than one feature of the application has to be checked, this TC should be split on as many TCs or as many features as has to be checked.
3. Each TC is recommended to both start and finish from the execution of the function that puts this app in the initial stage. It is necessary to do because if a previous TC failed and did not return the app to it's initial stage, it will be done by the "put in the initial stage" function from which the next TC is being started.
4. Each TC has to report explicitly whether it passed or failed. In case of a failure, a value of real result can be reported (expected string vs. actual string, expected sum vs. actual sum, etc.).
5. Each TC has to return 0 or non-0 return code to the main module depending on the result of its execution.
6. Each user-defined function has to return 0 or non-0 return code to the calling module depending on the result of its execution. In case of a failure within this function, it should report about the failure prior to the return statement.
7. Each module (TC or a user-defined function), that is calling a user-defined function, should check whether completion of the function was successful or not. In the case of a function returning non-0 code in the calling module, this module should return control back to its "parent" module (for example, in the main module) with non-0 return code. Before returning control, a calling module may report about the failure; if it is doing so, the name of this module should be used
8. A main module that is calling TCs, should check whether completion of each TC was successful or not. In case of a failure of a TC, it should report about the failure
9. To minimize efforts and cost of scripts' maintenance, it is necessary to avoid hard coded data. Instead of hard coding, usage of variables is recommended. Values to the variables should be assigned at the beginning of Main module' or TCs' script.

Script Development

Upon completion of establishing design for scripts the Automation team will begin developing scripts for the test cases. The scripts will be written following "Best Practices" standards (see above) of the Automation team.
This ensures that when the scripts is handed over to QA Test team they will understand what the script is intended to do, who wrote it and when, setup conditions, variables definitions, names and locations of external files, etc. There will also be some Code Review by Automation Team Manager and/or other Automation Team member prior to Unit Test

Script Test

The script(s) will not be forwarded to QA Test team until they are thoroughly tested by person scripting on his/her machine running the AUT. The script(s) will be handed over to the QA Tester for testing. This will be performed on lab machine running same version of the AUT.
. NOTE: The naming convention for script will be the same as the name of the manual test case.

Script Release

Once the automation scripts are completed and thoroughly tested, they will be stored in a central repository.

Performance Test Scripting

As with any type of testing you need to know what requirements have to be met to

- decide what needs to be tested/measured
- define pass/fail criteria
- assess type of tools needed, if any, and
- how to design and execute the tool to determine pass/fails

	Description	*Objectives*
Load Testing	*To model the anticipated real-world performance of system's infrastructure over short period of time under normal conditions*	*To help assess whether a specific combination of SW and HW architecture meets performance requirements*
• *Stress Testing*	*To model the anticipated real-world performance of Web-site over short period of time under extreme conditions (maximum capacity during peak hours)*	*To determine whether a specific combination of SW and HW architecture has capacity to handle excessive amount of transactions during peak hours* *To determine what will happen if capacity is reached*
Capacity/Scalability Testing/Planning	*System's ability to increase capacity for future growth*	*To assess that performance/service is not disrupted or diminished with hardware upgrades*

- If the system does not fail during the stress test then the stress test should be considered deficient. Stress test cases should focus on transactions that require significant computation and/or manipulation of data in order to put excessive load on the system's input/output. Stress testing continues until a failure load is reached or it becomes obvious that the system can bear the heaviest loads generated, and that further increasing the load is wasted effort.

Performance as a Collaborative Effort

The performance of a system's hardware and software, no matter what the architecture, is not a one-team endeavor. Servers (Web, Application, Database), bandwidth, throughput, network traffic, etc. need to be monitored and tested. These activities are typically performed by different teams; Performance Test teams have the skill/experience for testing and test tools and Infrastructure or Server Support teams are chartered for administration and monitoring tasks. **This is a process that is a prime target for improvement** (i.e. cross-training of Performance Test and Infrastructure/Server Support teams).

The QA group should

- assess that these monitoring and testing activities happen
- ensure they are done at appropriate time in project and
- the proper tools are selected and implemented

The collaborative aspect to this type of testing is apparent by the nature of the parameters required for realistic test cases.
Knowing current, and estimating future, loads to the system/site is data that can be provided by server and trafficmonitors/tools.

The following are just a *few* parameters and measures of server load and User activities that performance-testing tools should replicate and measure:

- number of transactions per second (TPS)
- megabytes of network traffic per second
- number of concurrent users
- arrival rates on servers
- abandonment rates on servers
- usage models
- peak load hours
- Think time
-

The benefit of using tools in this effort is summarized in table below.

Performance Testing Manually	**Performance Testing with Tools**
Expensive – requiring large amounts of personnel and hardware	Reduces personnel requirements replacing human users with virtual users Reduces hardware requirements as multiple virtual users can exist on one machine
Complicated – coordinating and synchronizing scores of Users/Testers/Architects	Enables the control and coordination of all users from a single point of control
High degree of organization to collect and analyze meaningful test results	Automatic records performance results into meaningful graphs and reports
Repeatability of tests is limited	Automated scripts/scenarios can be repeated with no human error as often as, and whenever, needed