



## Automated Testing FAQ

### Q] What kind of testing is automation?

A] It must be fully understood from the beginning that test automation is not testing. It is software development! A script is a program that drives the system on the workstation to perform certain tasks. In some ways, an automated test is harder to design than a [regular] software program, because in addition to interacting with the operating system and current environment on the workstation, it must also interact with the Application Under Test (AUT.) The expectation must be that the AUT will not always operate correctly (or else why test?) Therefore, to be robust, extra programming must be performed on the script to allow for this uncertainty.

- As software development, as successful project must have all of the same characteristics as other SW projects.
  1. Requirements
  2. Design
  3. Code
  4. Test

### Q] What kind of testing is automation suited for?

A] Almost any kind of testing which may need to be run multiple times and/or on multiple machines:

- Regression
- System and Integration testing (2<sup>nd</sup> series)
- Smoke (Build Acceptance) Testing
- Multi-Platform testing
- Multi-Language Testing
- Focused testing during RAD (spiral) development for Windows applications
- Positive, Negative, and Edge conditions for controls
- Error handling
- Stress Testing (repeat test many times to check for memory / resource leaks)
- Load Testing

### Q] What kind of testing is automation not suited to?

A] Tests which are not generally repeated (payback point usually comes when test is played back 10-15 times or more.

- Investigative testing.
- Trivial applications (unless part of a major application suite.)
- Highly visual applications (usually)

### Q] Are there any useful by-products of automated testing?

A] For a testing department to be successful with automation, it must have certain mature test processes in place (i.e. formal test plans, formally designed testcases, etc.) If a testing department does not already have these in place, then the institution of automation will force it to adopt these processes; thus raising the test maturity level.



Q] **What are some the reasons not to automate?**

A] **Testing Maturity level too low.**

- No design documentation: not really sure what application is supposed to do
- No formal test plan: not really sure how we are going to approach the testing
- No formally designed testcases: just going to beat on the product in an ad hoc way!
- Application is very immature: it radically changes every build, and there is no product definition which points to the end product.
- Testing Budget is minimal; just enough to pay for a few part time testers at end of the cycle.
- Quality is not important.

Q] **What are some of the erroneous assumptions about automated testing?**

A] **Automation eliminates the need for manual testers.**

- Manual testers do not repeat the exact same actions through every test; therefore, they come upon problems that might not otherwise be caught.
- Manual Testers can explore areas using product knowledge and hunches.
- Manual Testers can react to subtle changes or Cues as they work with the SW and change their approach based on that.
- Manual Testers can target the specific areas just changed or fixed to leverage the testing time towards the most likely spots where bugs are found.

**Automation can do Investigative testing.**

- At today's level of automation, an automated script can only go over an area that the test scripiter guides it to. There is no artificial intelligence or random wandering available in automation yet. Therefore, by definition, the script, when played back, is going over ground already covered.

**Automated Tests run much quicker than manual tests.**

- This is partly true if the AUT (application under test) can handle the speed. Often, the application is limited in the speed that it can do processing. In addition, client / server is usually limited by network lag time. Longer automated tests usually require a higher level of investment in error handling, fault tolerance, infrastructure, etc. These factors tend to raise the cost of the testing.

**Repeating a test many times finds lots of bugs.**

- Not necessarily true. If a bug is to be found, the first run through (when the test is being created) often finds it. This usually occurs during the scripting phase of the automation cycle. Once the bug is fixed for that particular location, running it multiple times will not necessarily find more bugs. Repeating a test is useful if the data inputted to the AUT is different each run (data driven testing). It is also useful for testing resource and memory usage, where a leak is suspected. The automated scripts, therefore, are generally not used to find new and different bugs. They are useful to verify and validate that the areas already tested remain bug free (eliminate regression bugs) while the manual testing and new script development check other areas of the AUT for problems. It has been estimated that the scripting cycle finds 70% – 85% of the bugs that are discovered during the automation process.

**Automation is a complete testing strategy.**

- Absolutely not true at all. Automation is only a part of a complete strategy to leverage the test cycle into higher quality software. Done correctly, it can give very good benefit to cost return on the investment made. If a test department tries to make automation their only testing strategy, they are putting themselves and their applications at risk.



**Q] What are some of the problems with automation?**

**A] Not all bugs found are equal.**

- Often, a bug that is found is minor and can be ignored. An automated test script will generally catch this bug each time a script is run, causing false errors. To avoid this problem, scripts must have a higher level of programming, which raises the cost. A balance must be found to maximize ROI.

**Evaluation of the results of an automated test may be problematical.**

- Consider the case where an automated test failed during its execution. The following are all valid question:
  1. Do we know what failed?
  2. Was the failure caused by a non-trivial software error?
  3. Was the bug actually in the automated test itself?
  4. Was there no bug, just a timing glitch that was not foreseen?
  5. Was it a misunderstanding of the requirements?
  6. Was it an expected change in the AUT for which the test was not revised?
  7. Was the problem external to the test and AUT?

Note that the test script itself can automatically answer many of these questions, if it is programmed to provide fault tolerance. Programming, however, costs extra money. The tradeoff must always be made in SW development, cost vs. features.

**It is possible to write an automated test suite that can run for hours or days. Even if you write such a suite, can the AUT run that long without a normal failure? Is it a valid use of the AUT to run for hours or days?**

- The Windows environment tends to be brittle. Just during regular use, it needs to be booted every so often just to keep it coherent. (Win 95/98 more so than the Windows NT line.) It is possible to build fault tolerance and the ability to reboot the system, into the test suite, but it costs extra.

**If a bug is missed during the scripting, an automated test may treat that bug as a feature. Each time the test is run, the bug will show up but it will be ignored.**

- The automator must have a firm grasp of the AUT's requirements.

**Q] What are some of the problems with Automation tools?**

**A] Automation tools are software programs. Therefore, they always have bugs of their own which might cause problems.**

**Some applications do not run well with automation tools.**

- For example: if the AUT was written using the Smalltalk language, most automation tools will have great difficulties working with Record mode. The reasons are several; suffice it to say that the cost per script will be much larger than if the AUT was written [for example] with Visual Basic. This is because the script must be developed using high levels of programming rather than simple recording. The test can still be developed; it is just relatively expensive.
- Likewise, some applications might be written in a language that is standard, but include non-standard controls that the automation tool does not understand.

**In general, the automation tool can usually be made to work with the AUT no matter what it was written in; cost becomes the question.**

**Bleeding edge applications tend to cost more to automate. This is because there is a lag time between the time that an application feature is introduced and the time that an automation tool is**



**updated to work correctly with that feature. Therefore, to automate those AUTs, more programming must be completed.**

- Examples: Com, DCom, OLE, etc.

**The existence of the automation tool in the environment changes the environment. That means that there may be interactions between the automated tool and the AUT.**

- This is typically not a problem for most application testing; however, under certain circumstances, it must be taken into account.

**Q] What are the actual costs of automating my testing?**

**A] Assuming that the automation is to be a success, there are certain costs that must be calculated to determine the return on investment.**

- Original automation tool[s] cost.
- Testcase development (should be part of overall test planning.)
- Architecting the infrastructure for the testing (i.e. do functions need to be written to make the tool work with the AUT?)
- Creating the test scripts
- Testing the test scripts
- Fixing the test script bugs
- Documenting the test scripts
- Running the test scripts
- Diagnosing the results of test runs
- Administrative costs
  1. Deal with test tool package and vendor
  2. Librarian costs for tests and infrastructure functions
  3. Programming Standards and Guidelines
  4. Training
- Maintenance (change scripts as AUT changes)

**Q] What are some strategies for successful test automation?**

**A] When designing your Test Automation strategy, remember:**

***Test Automation is not testing, it is SW development.***

- Testers often have no SW development background. SW developers often have no test background. The skill sets for both tasks are largely disjoint. If the automation task is not run according to industry best development practices, according to historical analysis, it may fail.
- An experienced developer / automator should be in charge of the infrastructure building and architecting tasks. In addition, development standards and guidelines should be created and followed using best industry practices. Junior developers or technically qualified testers can write the actual scripts.
- An experienced tester should be in charge of developing the testcases to be automated, deciding:
  1. What is to be tested?
  2. In what order?
  3. With what verification needed?

**For most automation, a mature AUT is preferable. If the application is still changing radically, the automation costs for script rework could soar.**



**Mature testing process is essential. If there is no plan to the testing, any suite of tests that is developed is likely to be of marginal quality or worth.**

**Each automated test script should have an equivalent testcase that can be run manually in case of emergency. Ideally, those testcases should exist before the automation even starts.**

**Each testcase script should be totally documented with why it exists and how it works.**

- The reason for this requirement is not readily apparent. However, if you plan on using these automated tests over a period of time, it is essential. Two years down the road, the testers who are running the test may be completely different. How will they then know what they are testing? Is the test valid? Are there conditions under which the test passes but the AUT is broken?

**Don't play numbers games. Write tests that are designed to fail!**

- A test that tries hard to pass typically will. Remember, the question is not how many tests ran and passed; that thinking engenders a false sense of security. The question to be answered is, "Does the test suite we are running actually reveal any bugs which may have been injected during this round of development?"

## **Q] What are some initial targets for an automation program?**

**A] Usually, the initial automation effort should be aimed at a relatively small, focused area of need. This allows an initial automation process be developed freely, without the possibility of a major collapse of all testing if there are problems or delays. Since an initial small success can pave the way for later, larger successes; many successful automation programs have used this strategy. Some possibilities include:**

- A skeletal regression test suite to be used for smoke [build acceptance] testing.
- Compatibility testing of certain functionality (i.e. does this application run correctly when outputting to different printers, tape drives, etc.)
- Benchmarking (i.e. the SQL statement averaged 3 seconds to execute the last time through, 3.6 this time.)
- Multiple OS platform testing (again using a small but broad based set of regression tests.)
- Data intensive areas where a small number of data driven scripts can test a large number of data sets (i.e. SQL correctness.)
- High traffic areas where most interaction with an application occurs (80/20 rule.)
- Resource and memory testing (i.e. few, simple tests run in loop hundreds of times.)

**In working with this limited, focused set of tests, concentrate on building an infrastructure of functions which handle common tasks that occur. Examples:**

- If the application is always opening files, create an "OpenFile(xxx) function which handles all of the details of the open (i.e. handling errors, security log ins, etc.)
- If several sequential steps are often called, put them into a function that can be added to a script in one line.
- Create needed helper functions that are always used (String functions, logging routines, etc.)
- Create a Close routine that handles the case when a test fails and the AUT is left in an unknown position.

**Why are these routines useful? If (when!) your application changes, in the next iteration, the way that a file is opened, each script which you have will have to be modified. Unless, that is, all of the scripts use a function called OpenFile(xxx). In that case, the change only happens in one place (the function itself) and all scripts that use it remain functional. Once again, this is a strategy that must be viewed as a trade-off, between programming cost and eventual maintenance costs.**



**Build in error detection and logging to these functions. This simplifies the scripting process, because the functions are deemed to “do the correct thing” in case of an error. Any code that can be moved from the script to the infrastructure pays for itself many times over (the multiplier is the number of scripts which call the function.)**

**An additional task of this initial automation cycle is to capture the functionality of non-standard controls that are used by your application. Non-standard means that the tool does not understand the control or what it is meant to do. The tool only sees the widget as a generic object.**

- A typical example of this is a grid control that has various columnar data from a database in it. Often, the tool does not process these control types correctly; they are seen as a single big object and all interactions are seen as simple mouse clicks or drags. A set of functions should be created to make the grid appear as if the automation tool understands it. Examples:
  1. Select a given item from the grid.
  2. [Double] Click on a particular column.
  3. [Double] Click on a particular row.

Once these functions are captured, future use of this once problematic control becomes simple. If the user wants to click on the control, they simply add the function “GridClick(xxx,yyy,zzz)” to the script.