ISTQB® Advanced Test Automation Engineer

Objectives for Test Automation

1 Objectives for Test Automation	2 Preparing for Test Automation	3 The Generic Test Automation Architecture	4 Deployment Risks and Contingencies
5 Test Automation Reporting and Metrics	6 Transitioning Manual Testing	7 Verifying the TAS	8 Continuous Improvement

1.1

Contents

1.2

What is test automation?

- test automation:
  - covers many aspects of testing
  - often just used to mean: ‘test execution automation’
    - ► and mainly in this course
- is using tools to:
  - control and set up test preconditions
  - execute tests
  - compare actual outcomes with expected outcomes
- software used for testing that should be separate from the system under test (SUT) itself
- runs test cases consistently and repeatedly on different versions of the SUT

1.3

Process of designing testware

- test automation is not just running tests - it is a process of designing a test automation solution:
  - software
  - documentation
  - test cases
  - test environments
  - data
- - and other activities:
  - implementing automated tests
  - monitoring and controlling test execution
  - interpreting, reporting and logging the automated test results

1.4

Different approaches

- different approaches to test automation include:
  - testing through the public interfaces
    - ► to classes, modules or libraries of the SUT (API testing)
  - testing through the user interfaces of the SUT
    - ► (GUI testing or CLI testing)
  - testing through network protocols

1.5

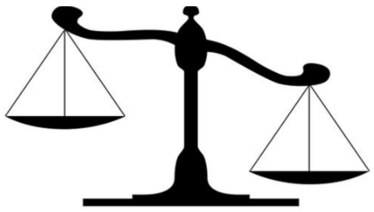
Objectives of test automation

- improving test efficiency
- providing wider coverage
- reducing the total test costs
- performing non-human-capable testing
- shortening the test period
- increasing the test frequency/reducing the time required for test cycles

1.6

Advantages of test automation

- more tests are run per build
- tests that cannot be done manually are enabled (real-time, remote, parallel tests)
- tests can be more complex
- tests run faster
- tests are less subject to operator error
- testers can concentrate on test analysis and design
  - adding greater value
- better co-operation with developers
- improved system reliability (e.g. repeatability,consistency)
- improved quality of test



1.7

Disadvantages of test automation

- additional costs are involved
- initial investment to setup TAS and train people
- requires additional technologies and tools
- team needs to have development and automation skills
- on-going TAS maintenance requirement
- can distract from testing objectives
  - e.g., focusing on automating test cases at the expense of executing tests
- tests can become more complex
- additional errors may be introduced by automation

1.8

Limitations of test automation

- not all manual tests can be automated
- the automation can only check:
  - machine-interpretable results
  - actual results that can be verified by an automated test oracle

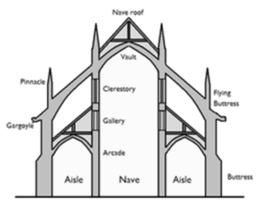
1.9

Contents

1.10

Success factor: Test Automation Architecture (TAA)

- Test Automation Architecture (TAA)
  - align with SUT architecture
  - supporting most important functional and non-functional requirements
    - typically:
      - maintainability,
      - performance
      - learnability
  - involve developers (who understand SUT architecture) in TAA design
- e.g.
  - don't use a TAA that is designed for command line driven applications on GUI applications



1.11

Success factor: SUT testability

- SUT Testability
  - SUT designed to support automated testing
  - e.g. GUI software
    - the SUT should decouple as much as possible the GUI interaction and data from the appearance of the graphical interface.
  - API testing
    - this could mean that more classes, modules or the command-line interface need to be exposed as public so that they can be tested.
- approach
  - target most easily tested parts of the SUT first

providing they are worthwhile testing

often the GUI changes even when the underlying functionality does not

for the duration of testing at least

1.12

Success factor: test automation strategy

- Test Automation Strategy
  - must address maintainability and consistency of the SUT
  - consider the costs, benefits and risks of applying the strategy to different parts of the SUT code
  - consider testing both the user interface and the API with automated test cases

e.g. guidelines/rules, scripting approach, test data format and volume, etc.

e.g. new code vs. old code, stable vs. unstable, complex vs. simple, etc.

to check the consistency of the results

1.13

Success factor: Test Automation Framework (TAF)

- TAF should be:
  - easy to use
  - maintainable
  - well documented
  - supporting a consistent approach to automating tests
    - ► consistency is king!
- usable and maintainable TAF
  - reporting facilities
  - enable easy troubleshooting
  - address the test environment appropriately
  - document the tests
  - trace the automated test
  - enable easy maintenance
  - keep the tests up-to-date
  - plan for deployment
  - retire tests as needed
  - monitor and restore the SUT

1.14

Achieving an easy to use and maintainable TAF (1)

- reporting facilities
  - about quality of the SUT
    - pass/fail/error/not run/aborted, statistical
  - for all stakeholders
- enable easy troubleshooting
  - to help locate defects in SUT, TAS, tests and test environment
- address the test environment appropriately
  - dedicated test environment
- document the tests
  - clearly state what each test does
- trace the automated test
  - allowing individual steps to be traced
- enable easy maintenance
  - keep maintenance costs low
  - tests analysable, changeable and expandable
  - testware reuse high

1.15

Achieving an easy to use and maintainable TAF (2)

- keep the tests up-to-date
  - fix tests that fail due to software updates
    - don't disable them!
- plan for deployment
  - ensure test scripts can be easily deployed, changed and redeployed
- retire tests as needed
  - when scripts are no longer useful
    - don't allow them to clutter
- monitor and restore the SUT
  - when SUT fails, TAF should have the capability to recover, skip the current case, and resume testing with the next case

1.16

Maintenance of automation code

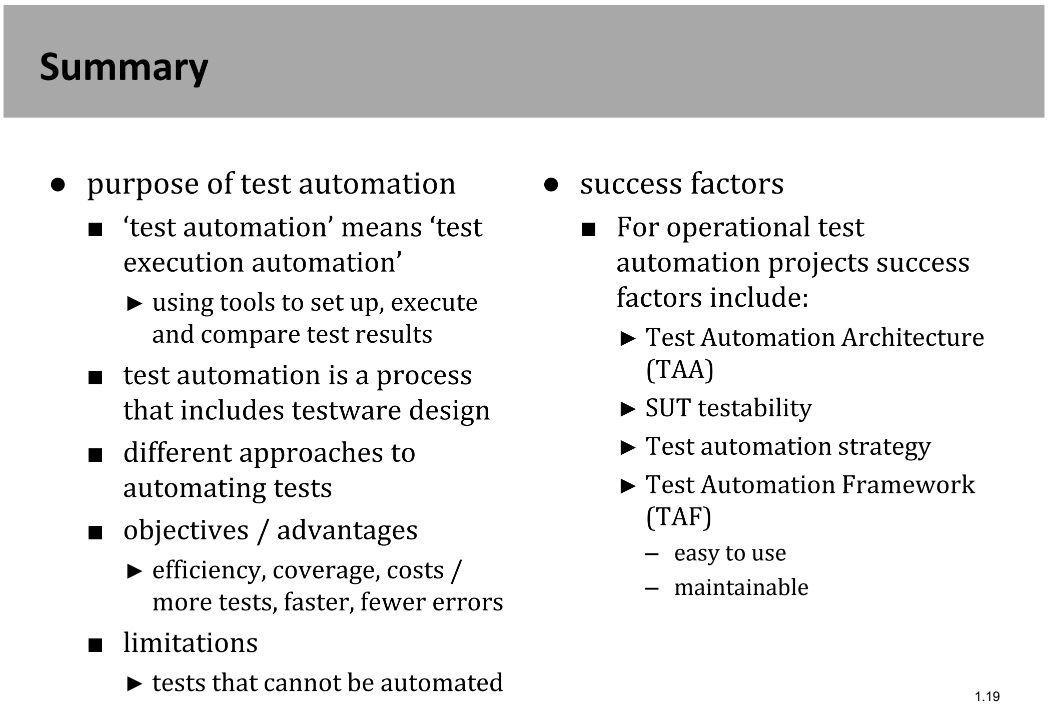
- automation code can be
  - complex to maintain
  - as much as SUT code
    - ► due to different test tools
    - ► different verification types
    - ► different testware artefacts
      - – e.g. test input, test oracles, test reports
- do not:
  - create interface sensitive code
  - create automation that is sensitive to data changes
  - dependency on data values
    - ► e.g. test input dependent on outputs of other tests
  - create automation environment sensitive to context
    - ► e.g. operating system data and time, localization parameters, other applications
      - – better to use stubs

1.17

Success factors

- the more success factors that are met, the more likely test automation will succeed
- not all factors are required (rarely achieved)
- before starting automation:
  - analyse chance of success by considering:
    - ► factors in place
    - ► factors missing
    - ► risks of chosen approach
    - ► project context

1.18



Summary

- purpose of test automation
  - ‘test automation’ means ‘test execution automation’
    - using tools to set up, execute and compare test results
  - test automation is a process that includes testware design
  - different approaches to automating tests
  - objectives / advantages
    - efficiency, coverage, costs / more tests, faster, fewer errors
  - limitations
    - tests that cannot be automated
- success factors
  - For operational test automation projects success factors include:
    - Test Automation Architecture (TAA)
    - SUT testability
    - Test automation strategy
    - Test Automation Framework (TAF)
      - easy to use
      - maintainable

1.19

Session 1

Introduction and Objectives

Terms
API testing, CLI testing, GUI testing, SUT, test automation framework, test automation strategy, test execution automation, test script, testware.

From the ISTQB Glossary

API testing: Testing performed by submitting commands to the software under test using programming interfaces of the application directly.

CLI testing: Testing performed by submitting commands to the software under test using a dedicated command-line interface.

GUI testing: Testing performed by interacting with the software under test via the graphical user interface.

SUT (system under test): See test object.

test automation framework: A tool that provides an environment for test automation. It usually includes a test harness and test libraries.

test automation strategy: A high-level plan to achieve long-term objectives of test automation under given boundary conditions.

test execution automation: The use of software, e.g., capture/playback tools, to control the execution of tests, the comparison of actual results to expected results, the setting up of test preconditions, and other test control and reporting functions.

test object: The component or system to be tested.

test script: Commonly used to refer to a test procedure specification, especially an automated one.

testware: Artefacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing.

1.1 Purpose of Test Automation

Learning Objective		
ALTA-E-1.1.1	K2	Explain the objectives, advantages, disadvantages, and limitations of test automation.

What is test automation?

Software test automation covers most aspects of software testing though the term is most commonly applied to the automation of test execution. Throughout this course, we use the term ‘test automation’ to mean test execution automation.

Test automation is using tools to perform one or more of the following tasks:

- control and set up test preconditions
- execute tests
- compare actual outcomes to predicted outcomes

The software used for testing (testware) should be separate from the system under test (SUT) itself. If the testware is not separate it can impact the behaviour of the SUT such that the test results are not representative of the experience users will have.

Process of designing testware

Test automation is expected to help run many test cases consistently and repeatedly on different versions of the SUT. But test automation is more than a mechanism for running a test suite without human interaction. It is a process of designing the testware, including the following:

- software
- documentation
- test cases
- test environments
- test data

The testware is necessary for the testing activities that include:

- implementing automated tests
- monitoring and controlling the execution of automated tests
- interpreting, reporting and logging the automated test results

Different approaches

There are different approaches to automating tests, each with their own advantages and disadvantages:

1. testing through the public interfaces to classes, modules or libraries of the SUT (API testing)
2. testing through the user interface of the SUT (GUI testing or CLI testing)
3. testing through the network protocol

Objectives of test automation

Objectives of test automation include:

- Improving test efficiency
- Providing wider coverage
- Reducing the total test cost
- Performing non-human-capable testing
- Shortening the test period

- Increasing the test frequency/reducing the time required for test cycles

Advantages of test automation

Advantages of test automation include:

- More tests are run per build
- Tests that cannot be done manually are enabled (real-time, remote, parallel tests)
- Tests can be more complex
- Tests run faster
- Tests are less subject to operator error
- More effective and efficient use of testers
- Better co-operation with developers
- Improved system reliability (e.g. repeatability, consistency)
- Improved quality of tests

The advantages experienced by an organisation using test automation will primarily be aligned to the objectives that they have planned to achieve. For example, where an objective for test automation is ‘shortening the test period’, this is likely to be achieved because of the advantage that ‘tests run faster’ and ‘tests are less subject to operator error’.

Often an organisation will also benefit from test automation in other ways (sometimes unexpected ways). For example, the advantage ‘better co-operation with developers’ is rarely planned but often occurs because of the test automation bringing developers and testers to work more closely together.

Disadvantages of test automation

Unfortunately, test automation does also have disadvantages, though these are usually outweighed by the advantages, at least for successful automation projects. Some of the more common disadvantages of test automation include:

- Additional costs are involved
- Initial investment to setup TAS
- Requires additional technologies
- Team needs to have development and automation skills
- On-going TAS maintenance requirement
- Can distract from testing objectives, e.g., focusing on automating tests cases at the expense of executing tests
- Tests can become more complex
- Additional errors may be introduced by automation

Although previously we stated the advantage automation brings us faster test execution, we need to consider the time required to develop the automated test. For example, if a complex manual test takes 3 weeks to automate, the full automation lifecycle for this test includes development and execution. Adding 3 weeks (of development time) to a shortened execution time reduces the perceived efficiency, at least for the first automated iteration. In another example, the test team needing to have development and automation skills is a disadvantage where these skills are not present in the test team. However, introducing the appropriate training into the test team can lead to ‘better co-operation with developers’.

Limitations of test automation

Limitations of test automation include:

- Not all manual tests can be automated

  For example, tests for certain aspects of usability such as attractiveness cannot be automated.

- The automation can only check machine-interpretable results

  For example, some tests will produce output files. Checking these files against a known baseline will tell us if they match. However, understanding why they don't match is not necessarily something that the automation can assist with. The automation can only check actual results that can be verified by an automated test oracle

  For example, automated verification of automated tests requires a set of specific checks or comparisons with known or expected values or rules. Without this oracle automated verification may be limited to checking the SUT is still running.

1.2 Success Factors in Test Automation

Learning Objective

ALTA-E-1.2.1	K2	Identify technical success factors of a test automation project.

The following success factors apply to test automation projects that are in operation and therefore the focus is on influences that impact on the long-term success of the project. Factors influencing the success of test automation projects at the pilot stage are not considered here.

Not achieving one or more of these success factors does not necessarily imply a test automation project will fail but it will be less likely to achieve its full potential. Similar, a test automation project that does achieve all these success factors is not guaranteed to succeed.

Major success factors for test automation include the following:

Test Automation Architecture (TAA)

The Test Automation Architecture (TAA) should be closely aligned with the architecture of a software product. It should be clear which functional and non-functional requirements the architecture is to support. Typically, this will be the most important requirements.

For example, when creating a TAA to support testing on a mobile platform, special care should be taken to ensure that all targeted mobile devices are taken into consideration. This initially may include the most popular telephones. The TAA should have the flexibility over time to add functionality into the overall architecture that also supports tablets or other mobile devices that are not phones. In that way, the TAA grows with the project needs.

Often the TAA is designed for maintainability, performance and learnability. (See ISO/IEC 25000:2014 for details of these and other non-functional characteristics.) When designing the TAA it is helpful to involve software engineers who understand the architecture of the SUT.

SUT Testability

The SUT needs to be designed for testability that supports automated testing. In the case of GUI testing, this could mean that the SUT should decouple as much as possible the GUI interaction and data from the appearance of the graphical interface. In the case of API testing, this could mean that more classes, modules or the command-line interface need to be exposed as public so that they can be tested.

The testable parts of the SUT should be targeted first. Generally, a key factor in the success of test automation lies in the ease of implementing automated test scripts. With this goal in mind, and to provide a successful proof of concept (POC), the Test Automation Engineer (TAE) needs to identify modules or components of the SUT that are easily tested with automation and start from there.

This is recommended even when the functionality targeted for automation is of lower priority for testing. For example, automating tests for the browsing functionality of an online store is likely to be easier than automating tests for the payment functionality. This is because the payment functionality is likely to be more complex functionality and involve third-party components. However, the payment functionality may be more critical and higher priority for testing. The recommendation remains: automate what is easier to implement first.

Test Automation Strategy

A practical and consistent test automation strategy that addresses maintainability and consistency of the SUT.

It may not be possible to apply the test automation strategy in the same way to both old and new parts of the SUT. When creating the automation strategy, consider the costs, benefits and risks of applying it to different parts of the SUT code.

For example, the maintenance effort required for automated tests for one part of the SUT may differ significantly from the maintenance effort required for some other part of the SUT. This can be for various reasons including different implementation approaches, different functionality and different quality characteristics (such as stability).

Inconsistent implementation approaches across the SUT will typically mean much more effort has to go into implementing test automation than would have been required otherwise. In extreme situations, it may not be viable (or at least less beneficial) to automate tests across the whole of the SUT because of the cost of automating tests for each part. The test automation strategy should therefore have a means of selecting which parts of the SUT can be automated with the least effort and the greatest benefit.

Consideration should be given to testing both the user interface and the API with automated test cases to check the consistency of the results.

However, often what happens is that testers assigned to automation will not take the time to do proper planning and feel the need to start automating scripts immediately to meet management’s expectations. Take, for example, an application that continues to show errors in production and where there is a heightened requirement to address quality issues before release. By starting to automate without an understanding of the “big picture” requirements will inevitably result in a compromised solution, and one that often ends up being scrapped.

Test Automation Framework (TAF)

A test automation framework (TAF) that is easy to use, well documented and maintainable, supports a consistent approach to automating tests.

Achieving an easy to use and maintainable TAF

To establish an easy to use and maintainable TAF, the following should be done:

- Implement reporting facilities: The test reports should provide information (pass/fail/error/not run/aborted, statistical, etc.) about the quality of the SUT. Reporting should provide the information for the involved testers, test managers, developers, project managers and other stakeholders to obtain an overview of the quality.
- Enable easy troubleshooting: In addition to the test execution and logging, the TAF should provide an easy way to troubleshoot failing tests. The test can fail due to:
  - a failure of the SUT
  - a failure of the TAS
  - a problem with the test or the test environment.
- Address the test environment appropriately: Test tools are dependent upon consistency in the test environment. Having a dedicated test environment is necessary in automated testing. If there is no control of the test environment and test data, the setup for tests may not meet the requirements for test execution and it is likely to produce false execution results.
- Document the automated test cases: The goals for test automation should be clear, e.g., which parts of the application are to be tested, to what degree, and which attributes are to be tested (functional and non-functional). This must be clearly described and documented.

- Trace the automated test: TAF shall support tracing for the test automation engineer to trace individual steps to test cases.
- Enable easy maintenance: Ideally, the automated test cases should be easily maintained so that maintenance will not consume a significant part of the test automation effort. In addition, the maintenance effort needs to be in proportion to the scale of the changes made to the SUT. To do this, the cases must be easily analysable, changeable and expandable. Furthermore, automated testware reuse should be high to minimize the number of items requiring changes.
- Keep the automated tests up-to-date: when new or changed requirements cause tests or entire test suites to fail, do not disable the failed tests – fix them.
- Plan for deployment: Make sure that test scripts can be easily deployed, changed and redeployed.
- Retire tests as needed: Make sure that automated test scripts can be easily retired if they are no longer useful or necessary.
- Monitor and restore the SUT: In real practice, to continuously run a test case or set of test cases, the SUT must be monitored continuously. If the SUT encounters a fatal error (such as a crash), the TAF must have the capability to recover, skip the current case, and resume testing with the next case.

Maintenance of automation code

The test automation code can be complex to maintain. It is not unusual to have as much code for testing as the code for the SUT. Therefore, it is of utmost importance that the test code be maintainable. This is due to the different test tools being used, the different types of verification that are used and the different testware artefacts that are to be maintained (such as test input data, test oracles, test reports).

With these maintenance considerations in mind, in addition to the important items that should be done, there is a few that should not be done, as follows:

- Do not create code that is sensitive to the interface (i.e., it would be affected by changes in the graphical interface or in non-essential parts of the API).
- Do not create test automation that is sensitive to data changes or has a high dependency on particular data values (e.g., test input depending on other test outputs).
- Do not create an automation environment that is sensitive to the context (e.g., operating system date and time, operating system localization parameters or the contents of another application). In this case, it is better to use test stubs as necessary so the environment can be controlled.

The more success factors that are met, the more likely the test automation project will succeed. Not all factors are required, and in practice rarely are all factors met. Before starting the test automation project, it is important to analyse the chance of success for the project by considering the factors in place and the factors missing, keeping risks of the chosen approach in mind as well as the project context. Once the TAA is in place, it is important to investigate which items are missing or still need work.

1.3 Summary

Purpose of Test Automation

Remember that in this course the term ‘test automation’ means ‘test execution automation’, using tools to perform one or more of the tasks: control and set up test preconditions, execute tests, compare actual with expected results.

Test automation is a process that includes designing the testware (software, documentation, test cases, environments, test data).

There are different approaches to automating tests: using public interfaces, user interface and a network protocol.

There are many and varied objectives for test automation such as improving test efficiency, increasing test coverage and reducing total testing costs. Some of the common advantages of test automation include running more tests per build, test run faster, fewer human errors during test execution and more efficient use of human testers.

Test automation has its limitations, including tests that cannot be automated and test outcomes that cannot be verified by a tool.

Success Factors in Test Automation

For test automation projects that are in operation (i.e. not the pilot stage) success factors include the following:

- Test Automation Architecture (TAA)
- SUT testability
- Test automation strategy
- Test Automation Framework (TAF)

The TAF should be easy to use and maintain.

To establish an easy to use and maintainable TAF, the following should be done:

- Implement reporting facilities
- Enable easy troubleshooting
- Address the test environment appropriately
- Document the automated test cases
- Trace the automated test
- Enable easy maintenance
- Keep the automated tests up-to-date
- Plan for deployment
- Retire tests as needed
- Monitor and restore the SUT