Certified Tester
Test Automation Strategy
Syllabus

Version 1.0

International Software Testing Qualifications Board

Copyright Notice

Revision History

Version	Date	Remarks
Syllabus v1.0	2024/05/03	CT-TAS v1.0 GA Release

Table of Contents

Acknowledgements

This document was formally released by the General Assembly of the ISTQB® on May 3, 2024.

It was produced by the Test Automation Task Force of the Specialist Working Group from the International Software Testing Qualifications Board: Graham Bath (Specialist Working Group Chair) Andrew Pollner (Specialist Working Group Vice Chair and Test Automation Task Force Chair), Péter Földházi, Patrick Quilter, Gergely Ágnecz, László Szikszai. Test Automation Task Force reviewers included: Armin Beer, Armin Born, Geza Bujdoso, Renzo Cerquozzi, Jan Giesen, Arnika Hryszko, Kari Kakkonen, Gary Mogyorodi, Chris van Bael, Carsten Weise, Marc-Florian Wendland.

Technical Reviewer: Gary Mogyorodi

The following persons participated in the reviewing, commenting, and balloting of this syllabus:

Horváth Ágota, Laura Albert, Prasunkumar Banerjee, Jürgen Beniermann, Armin Born, Piet de Roo, Nicola De Rosa, Dingguofu Ding Guofu, Elizabeta Fourneret, Jan Giesen, Erik Haartmans, Matthias Hamburg, Tobias Horn, Mattijs Kemmink, Ilia Kulakov, Ashish Kumar, Vincenzo Marrazzo, Marton Matyas, Patricia McQuaid, Smitha Mohandas, Ingvar Nordström, Sreeja Padmakumari, Nishan Portoyan, Meile Posthuma, Swapnil Shah, Péter Sótér, Szilard Szell, Richard Taylor, Giancarlo Tomasig, Chris Van Bael, Daniel Van der Zwan, Carsten Weise, Marc-Florian Wendland, Claude Zhang.

0 Introduction

0.1 Purpose of this Syllabus

This syllabus forms the basis for the International Software Testing Qualification for the Test Automation Strategy Specialist qualification. The ISTQB® provides this syllabus as follows:

1. To member boards, to translate into their local language and to accredit training providers. Member boards may adapt the syllabus to their particular language needs and modify the references to adapt to their local publications.
2. To certification bodies, to derive examination questions in their local language adapted to the learning objectives for this syllabus.
3. To training providers, to produce courseware and determine appropriate teaching methods.
4. To certification candidates, to prepare for the certification exam (either as part of a training course or independently).
5. To the international software and systems engineering community, to advance the profession of software and systems testing, and as a basis for books and articles.

0.2 Test Automation Strategy in Software Testing

The Test Automation Strategy Specialist qualification is aimed at anyone involved in software testing and test automation. This includes people in roles such as testers, test analysts, test automation engineers, test consultants, test architects, test managers, and software developers. This Specialist qualification is also appropriate for anyone who wants a basic understanding of test automation, such as project managers, quality managers, software development managers, business analysts, IT directors and management consultants.

The Specialist Test Automation Strategy (CT-TAS) syllabus presents multiple factors that come into play when planning for test automation within an organization. Technical engineering implementation aspects of test automation methods and best practices are not in scope as they are covered in the separate ISTQB CTAL-TAE syllabus.

The Test Automation Strategy addresses test automation needs beyond those that are technical tool implementation and integration challenges. A strategic view of test automation provides a vision of implementation across projects within an organization in a systematic and consistent manner that ultimately demonstrates value to the organization.

0.3 Career Path for Testers and Test Automation Engineers

The ISTQB® scheme provides support for testing professionals at all stages of their careers offering both breadth and depth of knowledge. Individuals who achieve the ISTQB® Test Automation Strategy

Specialist certification may also be interested in the Test Automation Engineering (CTAL-TAE) qualification.

Individuals who achieve the ISTQB® Certified Tester Test Automation Strategy Specialist certification may also be interested in the Core Advanced Levels (Test Analyst, Technical Test Analyst, and Test Manager) and thereafter Expert Level (Test Management or Improving the Test Process). Anyone seeking to develop skills in testing practices in an Agile environment area could consider the Agile Technical Tester or Agile Test Leadership at Scale certifications. The Specialist stream offers a deep dive into areas that have specific test approaches and test activities e.g., in Test Automation Engineering, Performance Testing, Security Testing, AI Testing, and Mobile Application Testing, or where domain specific know-how is required (e.g., Automotive Software Testing or Game Testing). Please visit www.istqb.org for the latest information of ISTQB´s Certified Tester Scheme.

0.4 Business Outcomes

This section lists the Business Outcomes expected of a candidate who has achieved the Test Automation Strategy Specialist certification.

A Certified Tester Test Automation Strategy Specialist can…

TAS-B01	Understand factors from software and systems that influence the success of test automation
TAS-B02	Identify costs and risks of implementing a test automation solution
TAS-B03	Understand the roles and responsibilities of people who are contributing to test automation
TAS-B04	Plan for the integration of test automation across test levels
TAS-B05	Identify strategic considerations for test automation implementation in different software development lifecycle models
TAS-B06	Understand applicability and viability of test automation
TAS-B07	Plan test automation solutions that meet the organizational needs
TAS-B08	Understand test automation deployment strategies
TAS-B09	Understand test automation dependencies within the test environment
TAS-B10	Understand costs for setting up and maintaining test automation
TAS-B11	Learn which test automation metrics help drive decision-making
TAS-B12	Identify ways in which test automation brings value to the project and organization
TAS-B13	Identify test automation reporting requirements to address stakeholder needs
TAS-B14	Define transitioning activities from manual testing to test automation
TAS-B15	Define a test automation strategy that ensures projects share assets and methods to ensure consistent implementation across the organization

0.5 Examinable Learning Objectives and Cognitive Level of Knowledge

Learning objectives support business outcomes and are used to create the Certified Tester Test Automation Strategy Specialist exams.

In general, all contents of this syllabus are examinable at a K2 and K3 levels, except for the Introduction and Appendices. That is, the candidate may be asked to recognize, remember, or recall a keyword or concept mentioned in any of the six chapters. The specific learning objectives levels are shown at the beginning of each chapter, and classified as follows:

- K2: Understand
- K3: Apply

Further details and examples of learning objectives are given in Appendix A.

All terms listed as keywords just below chapter headings shall be remembered, even if not explicitly mentioned in the learning objectives.

0.6 The Test Automation Strategy Specialist Certificate Exam

The Test Automation Strategy Specialist Certificate exam will be based on this syllabus. Answers to exam questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the Introduction and Appendices. Standards and books are included as references, but their content is not examinable, beyond what is summarized in the syllabus itself from such standards and books.

Refer to the Exam Structures and Rules v1.1 Compatible with Syllabus Foundation and Advanced Levels and Specialist Modules document for further details regarding the Test Automation Strategy Syllabus Certificate exam.

The entry criterion for taking the Test Automation Strategy certification exam is that candidates have an interest in software testing and test automation. However, it is strongly recommended that candidates also:

- Have at least a minimal background in software and systems development and leadership for implementation of technology into the enterprise and experience as a senior test engineer, test lead or as a software developer
- Take a course that has been accredited to ISTQB standards (by one of the ISTQB-recognized member boards).

Entry Requirement Note: The ISTQB® Foundation Level certificate shall be obtained before taking the Test Automation Strategy Syllabus Specialist certification exam.

0.7 Accreditation

An ISTQB® Member Board may accredit training providers whose course material follows this syllabus. Training providers should obtain accreditation guidelines from the Member Board or body that performs the accreditation. An accredited course is recognized as conforming to this syllabus and is allowed to have an ISTQB® exam as part of the course.

The accreditation guidelines for this syllabus follow the general Accreditation Guidelines published by the Processes Management and Compliance Working Group.

0.8 Handling of Standards

There are standards referenced in the Test Automation Strategy Syllabus (e.g., IEEE, ISO, etc.). The purpose of these references is to provide a framework (as in the references to ISO 25010 regarding quality characteristics) or to provide a source of additional information if desired by the reader. Please note that the syllabus uses the standard documents as reference. The standards documents are not intended for examination. Refer to Chapter 7 References for more information on Standards.

0.9 Keeping It Current

The software industry changes rapidly. To deal with these changes and to provide the stakeholders with access to relevant and current information, the ISTQB working groups have created links on the www.istqb.org website, which refer to supporting documents and changes to standards. This information is not examinable under the Test Automation Strategy Specialist syllabus.

0.10 Level of Detail

The level of detail in this syllabus allows internationally consistent courses and exams. To achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the Test Automation Strategy Specialist
- A list of terms that students must be able to recall
- Learning objectives for each knowledge area, describing the cognitive learning outcome to be achieved
- A description of the key concepts, including references to sources such as accepted literature or standards

The syllabus content is not a description of the entire knowledge area of software testing; it reflects the level of detail to be covered in Test Automation Strategy Specialist training courses. It focuses on test concepts and techniques that can apply to all software projects, including those following Agile methods. This syllabus does not contain any specific learning objectives related to Agile testing, but it does discuss how these concepts apply in Agile projects and other types of projects.

0.11 How this Syllabus is Organized

There are six chapters with examinable content. The top-level heading for each chapter specifies the time for the chapter; timing is not provided below chapter level. For accredited training courses, the syllabus requires a minimum of 12.75 hours of instruction, distributed across the six chapters as follows:

- Chapter 1: 45 minutes – Introduction and Objectives for Test Automation Strategy
  - The tester understands the concepts of test automation and learns the selection criteria for candidate projects
  - The tester understands the factors that define a successful test automation implementation
- Chapter 2: 60 minutes – Test Automation Resources
  - The tester learns the different solutions that are available for test automation and the relative investment for each
  - Software licensing for test automation tools is covered
  - The testers understand the skills needed for test automation
- Chapter 3: 225 minutes – Preparing for Test Automation
  - The tester learns how test automation is used across and within test levels
  - Test automation strategies to adequately distribute testing and achieve shift left and shift right are covered
  - The tester learns how test automation supports legacy and Agile projects
  - Test automation within DevOps and continuous testing practices is covered
  - The tester understands how to define criteria for use of test automation, including tests most suitable for test automation
- Chapter 4: 135 minutes – Organizational Deployment and Release Strategies for Test Automation
  - The tester learns how test automation can improve time to market
  - The tester understands how to develop operationally relevant automated tests and reporting of defects
  - Test automation deployment strategy and risk mitigation is covered
  - The tester learns about the test automation environment and its dependencies
  - Integration of test automation and test data to a system under test is covered

- Chapter 5: 150 minutes – Test Automation Impact Analysis
  - Test automation metrics and reporting to help inform decisions is covered
  - A tester learns how to perform a return on investment for test automation
  - Objectives for an organization and a project to use test automation is covered
  - The tester learns how to analyze test reports and inform decision makers in a clear and understandable way
- Chapter 6: 150 minutes – Implementation and Improvement Strategies for Test Automation
  - The tester learns how to transition from manual to test automation and to continuous testing
  - Evaluating test automation for continuous improvement is covered

1 Introduction and Objectives for Test Automation Strategy – 45 minutes (K2)

Keywords

test automation architecture, test automation framework, test automation strategy

Learning Objectives for Chapter 1:

1.1 Success Factors of a Test Automation Project

CT-TAS-1.1.1 (K2) Explain the objectives and relevance of test automation

CT-TAS-1.1.2 (K2) Identify technical success factors of a test automation project

CT-TAS-1.1.3 (K2) Summarize appropriate investment criteria in selecting candidate projects for test automation

1.1 Success Factors of a Test Automation Project

1.1.1 Define Goals & Objectives for a Test Automation Strategy

When defining the strategy of a test automation project, the following need to be addressed:

- Defining the intended purpose
- Identifying risks
- Defining the scope
- Identifying the stakeholders involved
- Selecting the tools for automation
- Designing the test automation architecture (TAA)
- Identifying environments

Objectives of test automation may include:

- Improved test efficiency
- Broader and deeper coverage provided
- Improved overall quality of the SUT
- Reduced total cost and time to market
- Performed tests that manual testers cannot do
- Reduced test execution time
- Increased test frequency

Since Agile software development provides faster deployment cycles, and cloud applications are more widespread, development requires earlier and quicker feedback about the quality of the SUT. As a result of this shift, test automation has more focus and relevance in modern projects, given its nature and test objectives.

1.1.2 Identify Technical Success Factors of a Test Automation Project

The following success factors apply to test automation projects that focus on factors that impact the long-term success of a project. The use of a pilot project helps determine the tools and viability of selected technologies.

Success factors for test automation include the following:

- SUT testability
  - Enables test automation to access SUT interfaces
- Defined test automation strategy
  - The strategy needs to be applicable and customizable; its goals need to be achievable within time and cost constraints, and it needs to be kept up-to-date
- TAA
  - Clarity of what and how to implement
  - For additional information, see the CTAL-TAE Syllabus, section 3.1.1

- Test automation framework (TAF)
  - TAF that is easy to use, well documented and maintainable, supports a consistent approach to automating tests. For additional information, see the CTAL-TAE Syllabus, section 3.1.3.
  - Defined and implemented test reporting
  - Easy troubleshooting
  - Appropriate test environment
  - Documented automated test cases
  - Traceable automated tests to test case definitions and requirements
  - Easy maintenance
  - Up-to-date automated tests
  - A clear deployment plan
  - A clear test execution plan
  - Tests retired as needed
  - Effective exception handling

Before starting the test automation project, it is important to analyze the chance of success for the project by considering the factors in place and the factors missing, keeping risks of the chosen test approach in mind as well as the project context. Not all factors are required, and in practice, rarely are all factors met.

1.1.3 Summarize Appropriate Investment Criteria in Selecting Candidate Projects for Test Automation

Setting up test automation on a project means investment and, in most cases, significant cost. This should be taken into consideration, together with the nature of the project, and whether test automation should be used on the project.

Consider the following investment criteria before introducing test automation:

- Cost of introduction: Besides the work needed to set up test automation, additional costs will be required for the project as there may be a need to hire new test automation engineers (TAEs), buy new hardware, or set up training
- Current phase in the project's software development lifecycle (SDLC): It is better to start as early as possible so that test automation can bring more value sooner
- Expected/planned duration of the project/software development: For a shorter project, there may not be enough resources to start, or time left for test automation to bring value
- Maintenance cost: In the case of starting from scratch, setting up a test automation solution (TAS) will take time along with the need to be maintained

If the investment of introducing test automation on a project is acceptable, preparation for test automation can begin. This includes selecting the right test approach and test automation tools. The primary responsibility for this process is a strategic role of a test architect or test manager, who has the necessary test automation knowledge to make relevant decisions.

2 Test Automation Resources – 60 minutes (K2)

Keywords

test automation engineer, test automation solution

Learning Objectives for Chapter 2:

2.1 Costs and Risks of Implementing a Test Automation Solution

CT-TAS-2.1.1 (K2) Compare alternative technical solutions with regard to cost of ownership

CT-TAS-2.1.2 (K2) Explain licensing model considerations for test automation tools

CT-TAS-2.1.3 (K2) Provide examples of factors to be considered when defining a test automation strategy

2.2 Roles and Responsibilities within Test Automation

CT-TAS-2.2.1 (K2) Summarize the roles and skills necessary for a successful test automation solution

2.1 Costs and Risks of Implementing a Test Automation Solution

2.1.1 Compare Alternative Technical Solutions with Regard to Cost of Ownership

In terms of ownership, a common approach is a customized in-house solution based on open-source or commercial tools. Other alternative approaches include non-customizable commercial solutions or signing a contract with an outsource company to get the work done by the TAE.

Creating all the TAS in-house ensures that all costs, resources, risks, and governance are within the organization (e.g., team members/developers, and hardware and software resources that are necessary for the actual solution). A key factor is that, by following this approach and allocating time to this activity, the TAS can be developed without a need for a contract with a vendor or outsource company, as every required TAE and their knowledge is available within the organization. However, to carry it out successfully, the organization needs to have the right TAEs, hired or trained, who can drive the development of the TAS.

Working with a vendor-based solution, ownership will be shared between the client and the vendor depending on the details of their contract. If there is already a test tool in the organization that has already been piloted and meets all the requirements, and there is no need for additional features in the test tool, this approach will be easier to adopt. The testers within the organization will be able to work productively once receiving the necessary training from the vendor, and usually there will be subject matter experts (SMEs) assigned as product owners inside the organization. The risk with this approach is that if any additional work needs to be done (e.g., fix tool defects, and additional feature requests) to the test tool, it will take time and require negotiations with the vendor to get it done on time and done in the right way, which can affect the work of the testers who are working with this test tool.

If the organization does not want to have the ownership of setting up a team, outsourcing is a recommended solution. In the case of working with outsource companies, the client does not have to hire or buy any actual hardware, software, or recruit employees with the required skill set. All the work and additional costs will be with the outsource company along with the ownership of the tools as well. Measurable expectations and metrics must be defined in the contract to make the progress transparent and visible. This approach is suggested in case the organization does not plan to invest efforts into hiring TAEs due to short project timelines, costs, or other considerations.

2.1.2 Explain Licensing Model Considerations for Test Automation Tools

Licensing is an important aspect to consider when test automation is established. Each licensing model mentioned below has a different impact on usability and cost factors e.g., costs of test execution, and difficulty to set up the development environment.

Licensing models include:

- **Open-source:** In many cases, organizations are using open-source tools to achieve their test objectives in test automation. The main reason for this is that there are no license costs or any maintenance fees for the use of the test tools. Many users and organizations contribute towards the development of open-source tools which makes information gathering and receiving support easier. Most open-source licenses also allow people to modify the tools if necessary or even republish them without significant restrictions.

- **License per user/machine:** Commercial tools often have licensing per user(s) or machine(s). In terms of costs, tools can be used efficiently when the organization knows the number of TAEs who will work on a given project for the long term. Often, the more licenses an organization orders, the better the pricing they receive.
- **Floating license:** This is a license that can be shared between multiple people in the organization to be used at different times. It can be very useful when there are many TAEs who work on different machines. The number of licenses is calculated based on concurrent use, not total number of TAEs or specific machines on which they run their tests.
- **Runtime license:** Many commercial tools have runtime licenses for executing test automation. This type of license occurs with cloud vendors that host test automation tools, multiple operating systems (OSs), and browser versions as a service. There are also cloud vendors that provide access to device farms that have a variety of mobile devices, platforms, versions, and cellular networks. Those that utilize these services are only charged for the time they use to execute tests, and therefore are charged on a runtime license basis.

2.1.3 Provide Examples of Factors to be Considered When Defining a Test Automation Strategy

Many factors can influence decisions about test automation implementation and its strategy.

The most crucial factors are:

- Time constraints
- Needed level of expertise and number of TAEs to develop the TAS
- Test hardware
- Test tool licenses
- Adaptability
- Maintenance
- Support for different platforms (e.g., Web, desktop and/or mobile)
- Continuous integration/continuous delivery (CI/CD) support
- Test management integration and test reporting

The first and most important cost factor is the timeline for the work that must be done and for the test automation itself.

From a timing perspective, if the deadline is short, an approach often used is to hire only a few skilled TAEs to create the TAS. In terms of a longer roadmap and schedule, the organization can decide on the number of required TAEs with more context. When the SUT grows, improves, and becomes more complex, a decision can be made to hire more TAEs to implement and maintain the TAS and the tests.

Other important cost factors are the tools and their licenses. The budget should contain the required test tools, hardware, and additional training for the TAEs. These factors have a strong connection to the integration to other tools and systems to do any additional non-testing functions like uploading test results to the test management system or triggering a build to the configuration management system. For these additional functions there are tools, either free or with fees, and libraries to use, but these must be considered during the creation of the test automation strategy and should be included in the budget.

It is always context dependent as to how many test environments and agents a development team has. The larger and more complex the SUT and release schedule is, the more resources the organization will require, which will increase the cost as well. A recent approach to limit resource costs use cloud providers and pay for on demand test hardware resources and runtime licenses which must be used carefully as it can backfire. Sometimes machines can be left running and become more expensive than on their own hardware. This approach can reduce the high costs of maintenance and operation for the TAE team.

2.2 Roles and Responsibilities within Test Automation

2.2.1 Summarize the Roles and Skills Necessary for a Successful Test Automation Solution

For a successful TAS, organizations require skilled TAEs who should have strong programming and technical architecture knowledge.

Depending on the size and maturity of the project, there should also be at least one strong SME (e.g., a test lead, architect, and business analyst) who understands the actual business domain and the test objectives. The role of this person is to help drive and create the concept, based on test objectives, and a roadmap for the actual TAS. Team management and soft skills will be needed to train, motivate, and build the team for the actual work. [CTEL-TM-MTT]

A TAE should have strong technical skills and knowledge about different SDLCs, and about the architecture of the SUT and its development environment. Apart from the technical aspects, a TAE should have the ability to cooperate with test analysts and other stakeholders about the objectives of test automation. Since exhaustive testing is not achievable, the same applies for test automation: 100% test automation coverage is not achievable. As time and effort are limited, the TAEs need to be able to prioritize the most impactful test conditions to cover from a business and investment perspective by contributing to risk assessments.

3 Preparing for Test Automation – 225 minutes (K3)

Keywords

API testing, component testing, contract testing, shift left, shift right, system under test, test automation approach, test condition, test double, test level, test pyramid

Learning Objectives for Chapter 3:

3.1 Integration Across Test Levels

CT-TAS-3.1.1 (K2) Differentiate between test automation distributions

CT-TAS-3.1.2 (K2) Select a test automation approach based on the system under test architecture

CT-TAS-3.1.3 (K3) Demonstrate ways to optimize test automation distribution to achieve shift left and shift right approaches

3.2 Strategic Considerations in Different Software Development Software Development Lifecycle Models

CT-TAS-3.2.1 (K2) Explain how test automation projects conform with legacy software development lifecycle models

CT-TAS-3.2.2 (K2) Explain how test automation projects conform with Agile software development best practices that support test automation

CT-TAS-3.2.3 (K3) Prepare for test automation projects to conform with DevOps best practices that support test automation in continuous testing

3.3 Application and Viability of Test Automation

CT-TAS-3.3.1 (K2) Explain criteria for determining the suitability of tests for test automation

CT-TAS-3.3.2 (K2) Identify challenges that only test automation can address

CT-TAS-3.3.3 (K2) Identify test conditions that are difficult to automate

3.1 Integration Across Test Levels

3.1.1 Differentiate Between Test Automation Distributions

Mike Cohn devised the original concept of a test automation pyramid describing three levels: unit (the ISTQB calls this component), service, and UI. Since then, many variations have appeared, and they all share the same basic principles of describing the role of test levels and distribution of test cases across those levels (see the ISTQB CTFL Syllabus, section 5.1.6 Test Pyramid).

The focus in Mike Cohn's original test automation pyramid is on what test types are carried out by the TAS. The service level can be broken down into three test types: component integration testing, contract testing and API testing.

Unit testing, (the ISTQB calls this component testing) is for validating individual components, focusing on the code quality. Component integration testing allows for validation of the user interface (UI) and API by leveraging test doubles, such as mocks and stubs. Contract testing enables validation of the contracts between services. API testing focuses on the functional validation of given services with real data through real service connections. UI testing is end-to-end testing of a system, interacting with its GUI.

It is helpful to draw the current state of testing as a baseline and the target state. This provides a clear understanding of what is missing or what is an acceptable level of testing. This will indicate the amount of testing carried out on each test level and whether it is manual or automated testing. The target state also depends on the timeline, and what is achievable within that time frame. If feasible to achieve, then the pyramid shape should be the target shape.

Examples of test distributions:

- **Pyramid:** This is a balanced distribution of testing with less testing carried out on the higher test levels and more on the lower test levels with stable and quicker tests. If the available resources and time frame allows it, then this is often the target state pyramid.
- **Ice cream cone:** This is the inverted version of the pyramid. There is a balanced amount of service testing, but testing is heavily dependent on finding the majority of defects in the UI test level, which is typically more costly to automate due to its complexity. Due to the lack of component testing, defects are found later in the SDLC.
- **Hourglass:** Testing is heavy on the highest and lowest test levels, and service level testing is mostly missing, resulting in integration defects. If the business logic is provided by APIs (i.e., services), then many of the UI tests can easily be moved to lower test levels.
- **Umbrella:** Testing is completely dependent on costly UI tests. This results in slow defect turnaround, costly maintenance of test cases and the TAS. If it is not technically possible to implement lower-level test cases, then moving away from the umbrella shape might not be achievable, and the focus should be on optimizing the UI test automation suite, reducing test execution time, and improving stability.

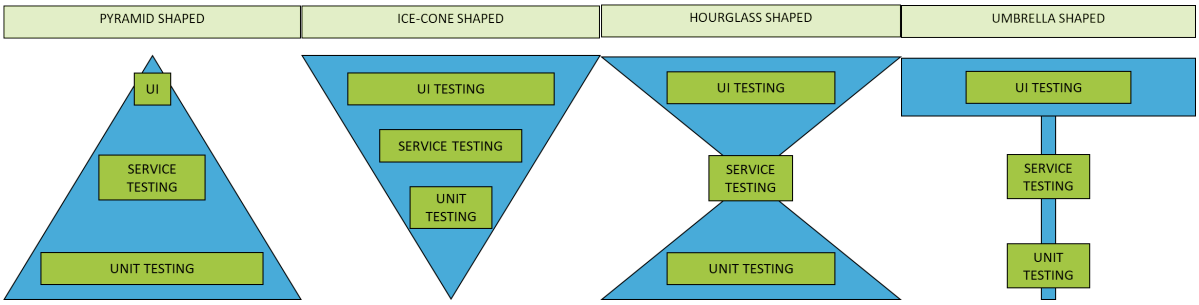

Figure 1: Examples of Test Distributions

3.1.2 Select a Test Automation Approach Based on the System Under Test Architecture

There are many ways one can define the test levels in a test strategy. Which one to choose is heavily dependent on several factors such as the organization's culture, overall maturity of software engineering, the SDLC followed and whether the SUT has a mainframe or microservices architecture.

Although the pyramid shape is considered as the ideal test distribution, it is not always achievable, or takes a longer time to transition to that final state. It is good practice to have a realistic shape as the target state such as transforming from the umbrella shape to the hourglass one. Once that is achieved, a new target state can be determined.

Choosing the appropriate test automation distribution for mainframe systems differs from modern software development and again depends on many factors as some of the levels might not be possible to automate at all. Access to mainframes is traditionally done through terminal emulation (i.e., green screens). Validation of the batch processes is possible but limited. Component testing is more difficult to carry out. As there are little to no microservices present, it is not always an option to validate data communication between interfaces with API testing. Testing through batch jobs, databases and a GUI is more prevalent.

Organizations that are modernizing their legacy solutions, slowly shifting towards a microservices architecture can introduce API testing and contract testing to the modernized parts of the system.

3.1.3 Demonstrate Ways to Optimize Test Automation Distribution to Achieve Shift Left and Shift Right Approaches

Once the current state distribution is identified and the target state distribution is selected, then a roadmap of improvements can be planned. This determines what to test with automation (i.e., the scope of test automation) and how to test (i.e., the test automation strategy). A prioritized backlog of items to implement and the entry criteria for test automation need to be determined.

In case of an unbalanced test automation distribution, it is recommended to go with a bottom-up approach. If code coverage is low, then it is a sign of a lack of component test cases, and additional component test cases need to be written. As a second step, the quality of the component tests needs to be reviewed and, if necessary, improved. Suggesting best practices, following a test-driven development technique, and holding test technique workshops can improve the quality of the test automation.

Introducing component testing for UI and API tests, leveraging test doubles (e.g., mocks, stubs) helps achieve a shift left from expensive, slow, and unreliable tests. Removing the reliance on real services and data improves the consistency of test execution and provides early feedback that is easy to integrate into CI/CD pipelines.

In recent years, contract testing has been increasingly prominent. By validating the contract between a provider and a consumer, it is easier to find root causes of defects earlier. Teams will not be reliant on API tests, nor UI tests to detect defects from underlying services, and instead can decrease the number of such test cases. Building a call graph for APIs is a clever way to track which APIs are connected to each other, and which ones are the consumers or the producers of a selected API. This allows better identification of potential pain points of a system.

While shift left is an approach that involves moving tests earlier in the SDLC, shift right moves tests later, when the SUT has been released, as a means to evaluate performance in a pre-production test environment or in production. By applying shift right, testers can monitor application and API performance while getting feedback from actual users. Although test automation is more prominent in a shift left approach, if it is combined with observability, then test automation can help in making decisions quicker on whether a full release can go ahead, or the release candidate needs to be rolled back.

Shift right testing aims to:

- Understand user preferences
- Support canary releases and dark launches to minimally disrupt user functionality
- Identify defects in production early
- Help expand the scope and use of test automation
- Increase coverage

3.2 Strategic Considerations in Different Software Development Lifecycle Models

3.2.1 Explain How Test Automation Projects Conform with Legacy Software Development Lifecycle Models

In the waterfall model, the testing phase comes after the requirement analysis, system design and implementation phases. Due to that strict order, test automation activities start later. Longer cycles between testing means fewer opportunities to leverage test automation, and the feedback from test automation usually comes too late, resulting in a lower return on investment (ROI).

In the V-model, all the test planning along with documentation preparation happens in early phases. As an example, the architecture design phase produces the integration test design, which sets up testing earlier than in the waterfall. Unfortunately, the actual implementation of a TAS still comes later in the SDLC, and although the ROI of test automation is higher compared to the waterfall model, it still falls behind modern Agile software development practices.

3.2.2 Explain How Test Automation Projects Conform with Agile Software Development Best Practices that Support Test Automation

One of the ideals of following Agile software development is to achieve in-sprint test automation. This means determining all the necessary test automation activities as part of the exit criteria for each of the user stories. It includes the test case definitions, implementation of automated test cases, updates to the TAF and in some cases integration to a CI/CD pipeline. Organizations that do not apply Agile practices correctly do not include an estimate of test effort and either administer them in a separate ticket or do not monitor them at all. By achieving in-sprint test automation, Agile teams ensure that they are ready to deploy the agreed scope within or by the end of each sprint. If a team is not mature from an Agile perspectives, then the first goal should be achieving in-sprint testing, while automation would lag behind by a sprint. From that, a team can work their way into achieving in-sprint test automation.

3.2.3 Prepare for Test Automation Projects to Conform with DevOps Best Practices to Achieve Continuous Testing

Agile software development is focused on how work is organized, while DevOps is responsible for the end-to-end delivery of software. This is achieved by automating build, integration, test, deployment, and production activities. This facilitates continuous testing through feedback loops that ensure continuous improvement.

Emphasis is more on implementing lower-level automated test cases including component, component integration and contract tests. Reducing reliance on real data and services, the tests will be executed in a shorter time. If feasible, test automation should be executed in the same pipeline where the SUT is built. Different test suites are triggered after each development and build phase (e.g., local, pull request, merge, and deploy).

If the testers have the capacity to build larger UI and API test suites, then those are executed separately to provide additional value. Manual testing efforts are suggested to focus on exploratory testing and end-user feedback, as a complementary activity to test automation.

3.3 Applicability and Viability of Test Automation

3.3.1 Explain Criteria for Determining the Suitability of Tests for Test Automation

Selecting test cases for test automation is usually done by a test analyst who either understands which test cases can be automated, or by a TAE who has the necessary domain knowledge to make such decisions.

The following are considered when selecting and prioritizing test cases for test automation:

- Is it technically possible to implement the test cases in an automated fashion?
- Are there any technical challenges that impact the delivery of automated test cases? Is the team prepared and well trained to do the implementation work?
- Does the coding effort provide an adequate ROI? (see section 5.1.1)
- Is there value in running the test cases frequently?
- Is it a functional or non-functional test? Is it part of the smoke test suite, regression test suite, or confirmation test suite?
- Is the test case repeatable?
- Is the test case easy to maintain when the SUT changes due to updates?
- Does the test case cover frequently used business workflows?
- Is there a functional overlap between tests that allow reusability of test steps and test data?

3.3.2 Identify Challenges that Only Test Automation Can Address

There are certain tests that can only be carried out with test automation. These include categories when:

1. Manual test execution time takes longer than what is adequate
2. Execution of test cases need to be synchronized
3. Test results need to be available in a pipeline
4. Large log files need to be parsed for defects
5. Precision in timing in tests is required
6. Test permutations are required across multiple OSs, browsers, devices, locations, or configurations
7. A large volume of test executions and/or data input is required to maximize coverage
8. Any non-functional testing that requires automated monitoring and analysis, or input from a huge amount of users such as stress testing or reliability testing

3.3.3 Identify Test Conditions that are Difficult to Automate

There are certain test conditions that make automating test cases a difficult challenge or even an impossible task. There are many real-life examples. Some of these are listed below:

- Validation of design requirements including the consistency of the UI through different platforms. The overall look and feel of software are subjective and require a human's review.
- Any test case that involves too much human interaction. In the finance sector, a good example would be a loan application. In this process there can be cases of certain regulations or conditions (e.g., not enough income, and incorrect personal details). In such situations, agents need to double check the actual loan application and make manual decisions or contact the customer.
- Technical difficulties blocking test automation activities, such as OS level restrictions. One example is native OS software which sends text messages to the users. Interactions or validations of those text messages are not possible with UI test automation tools, as the OS does not allow interactions outside of the target SUT.
- Test conditions with a lengthy time dependency would make test automation inefficient and is often difficult to properly set up compared to manual test execution. As an example, the tester needs to login to the SUT, refresh the home page content and wait two hours for the SUT to automatically logout due to a session timeout. If the elapsed time cannot be altered in any manual way (e.g., mocks, server-side responses, and OS level time changes), then implementing such test cases in an automated fashion is not advised.

4 Organizational Deployment and Release Strategies for Test Automation – 135 minutes (K2)

Keywords

component, confirmation testing, quality gate

Learning Objectives for Chapter 4:

4.1 Test Automation Solution Planning

CT-TAS-4.1.1 (K2) Identify ways how test automation supports shorter time to market

CT-TAS-4.1.2 (K2) Identify ways in which test automation helps verify reported defects according to requirements

CT-TAS-4.1.3 (K2) Define approaches that allow for the development of operationally relevant scenarios for test automation

4.2 Deployment Strategies

CT-TAS-4.2.1 (K2) Define a test automation deployment strategy

CT-TAS-4.2.2 (K2) Identify test automation risks in deployment

CT-TAS-4.2.3 (K2) Define approaches to mitigate test deployment risks

4.3 Dependencies within the Test Environment

CT-TAS-4.3.1 (K2) Define test automation components in the test environment

CT-TAS-4.3.2 (K2) Identify infrastructure components and dependencies of test automation

CT-TAS-4.3.3 (K2) Define test automation data and interface requirements for integration within the system under test

4.1 Test Automation Solution Planning

4.1.1 Identify ways how test automation supports shorter time to market

Test automation helps bring software to the market faster because it helps cut down on the test cycle time. Testing prior to a software release requires a thorough amount of verification and validation. This can be particularly significant during regression testing as this type of testing promotes reuse, driving down cost, and providing consistency between test executions.

Test automation helps decrease the manual test effort and provides quick feedback to developers while covering the same scope of testing. It also allows testing earlier in the software development lifecycle if component tests and component integration tests are automated, which are typically not performed in a manual fashion.

A quality gate is an enforced measure built into your process that the software needs to meet before it can proceed to the next phase. Setting up quality gates based on test automation allows an accelerated deployment process to a pre-production or to a production environment. By leveraging these quality gates, defects can be found earlier, which results in the time to market being shorter. Test execution time can be reduced by leveraging parallel test execution and following a shift left approach. A shift left approach promotes a culture of quality consciousness and encourages testing earlier in the software development lifecycle. This may include multiple independent test cases or cross-browser testing.

For additional information, see sections 3.1.3, 6.1.2, and 6.2.1.

4.1.2 Identify Ways in Which Test Automation Helps Verify Reported Defects

Confirmation testing performed following a code fix can address a reported defect. A tester typically follows the test steps necessary to replicate the defect to verify that the defect no longer exists.

Defects have a way of reintroducing themselves into subsequent releases (e.g., this may indicate a configuration management or code repository management problem) and therefore confirmation tests are suitable candidates for test automation and can be added to the existing regression test suite.

An automated confirmation test typically has a narrow scope of functionality. Implementation can occur at any point once a defect is reported and the test steps needed to replicate it are understood.

Tracking automated confirmation tests allows for reporting the time and the number of test cycles expended in resolving defects.

By verifying fixes across multiple platforms, devices, browsers, and OS versions with test automation, the amount of time spent on testing is greatly reduced.

4.1.3 Define Approaches that Allow for the Development of Operationally Relevant Scenarios for Test Automation

Operational acceptance testing assures software readiness for production systems and is typically conducted right before a release. This effort is meant to do a final validation of the systems, components, and other infrastructure of the SUT, and to test its readiness for production. This is necessary because despite a TAE's best efforts, there is no guarantee that the SUT will behave the same way outside of the test environment and in production.

A good operational test plan will include testing for reliability, testing for fault tolerance, integrity, and maintainability. Below are test automation approaches that can be used:

- Static code analysis – Automated tools that analyze code for security and vulnerabilities. Test results are stored for trend analysis over time.
- End-to-end testing – Automated test scripts that conduct end-to-end user scenarios and exercise all aspects of the test environment to uncover any defects.
- Failover testing – Automated test scripts that specifically test what happens when an application's hardware comes offline. Some examples include how does the application recover when physical servers, cloud servers, networks, computer disks and other hardware components malfunction. Test scripts are designed to measure success or failure of the SUT's ability to automatically recover and this is particularly important for organizations implementing chaos engineering.
- Backup and restore testing – Automated test scripts that test the success of making a backup of the current version and then rolling back to a previous point (i.e., an older version of the software)
- Performance efficiency testing (e.g., load testing) – Automated test scripts that can be used to simulate many users exercising the SUT at once. Test results are stored for comparison and trended over time.
- Operational documentation review – Automated test scripts can be used for this activity to compare versions of the SUT's documentation and signal to development teams whether an update is needed based on new features being added to a particular release.
- Security testing – Automated test scripts can be created in combination with industry standard security test tools to evaluate the SUT, in a static and dynamic manner. Test results are stored over time for comparison and trend analysis.
- Monitoring based on an organization's service level agreement – Automated test scripts used during the SDLC can be repurposed to monitor production operations and send alerts if an automated test fails. This is a proactive measure to catch production outages before real users experience them.

Test automation for operational software validation can provide immense benefits especially if the automated tests can be executed repeatedly and in multiple environments. Dedicated automated test suites can be created so that test results can be generated and compared with previous test executions. This ensures consistency when new versions of the SUT are released. A reliable automated test suite of operational specific test conditions can reduce costs over time.

4.2 Test Automation Deployment Strategies

4.2.1 Define a Test Automation Deployment Strategy

A good test automation deployment strategy will take into account, but not limited to, the overall test environment, the test tools that are available, access to the SUT, test script storage and other dependencies, and test data provisioning. With these high-level considerations in mind, a TAE can begin thinking about a strategy for developing and deploying the TAS.

Test environment – The TAEs should consider how they are going to access the SUT in the different test environments. When they start to develop their test automation testware, whether they are using a keyword-driven approach or another solution, they must consider how this solution will run in multiple test environments. For example, a test script should be developed in such a way that it can run in a test environment and in a pre-production environment with minimal changes. Typically, it is a matter of changing a uniform resource locator (URL) for a Web-based application and then the test script runs the same way regardless of which test environment it is in.

Tools – TAEs will also need to consider what tools they are using to build the TAS. If it is a commercial tool, chances are they will need to understand how it is licensed. The TAE may find that their test environment needs to have access to the tool's license server. This needs to be considered when the test script is intended to be used in multiple test environments. Just because the licensing server is available in the test environment does not necessarily mean it will be available in the pre-production environment.

Software access – Important considerations need to be understood on how to access the SUT. Test scripts can be designed to accept parameters so that specific users or credentials with special access can be quickly updated when the SUT end points change. This again becomes very important when moving into different test environments and the URL needs to change. In addition, it may be necessary to use different credentials (e.g., via user and test accounts, biometrics, and smart cards) depending on the pre-production environment. A good, automated test script design will include enough flexibility so that simple parameters can be set, and the test scripts executed as expected regardless of the test environment.

Test script storage – The TAE will want to decide on a central location to store and manage automated test scripts. A good strategy would incorporate a source code repository that utilizes configuration management. In this way, the test scripts can be accessible from multiple test environments as long as they have access to the source code repository and versions can be created for the specific version of the SUT. All configurations and dependencies can be saved in the same repository as the test scripts and provide optimal portability. In short, the TAS, TAF, and all test cases can be stored and managed within repositories.

Data provisioning – It will be important to understand if an automated test script is dependent on certain test data already existing in the test environment where the SUT is being tested. Good test script design will avoid static data (i.e., fixed data sets) as much as possible. However, there will be situations where it is not feasible. In these cases, the TAE will need to determine a solution to have the test data preloaded or use test automation to create setup test scripts that generate the necessary test data before the real test scripts run and test the SUT. There are pros and cons to both approaches. On the one hand it is convenient to have an administrator preload test data. However, the TAE then relies on another resource to have the availability to support them. Being self-sufficient with a setup test script removes the need to rely on another resource. However, it takes time to put these test scripts together and becomes another part of the solution that needs to be maintained.

By accounting for all of the items mentioned above, the test automation strategy will be able to provide proper planning and control of test automation activities.

4.2.2 Identify Test Automation Risks in Deployment

Technical issues can lead to product risks and project risks (for additional information, see the CTFL Syllabus, section 5.2.2). Typical technical issues include:

- Too many abstractions can lead to difficulty in understanding what the test automation code is really doing (e.g., with keywords in the keyword-driven approach)
- Test data tables can become too large/complex/cumbersome to migrate to other test environments resulting in inconsistent status outcomes
- Dependency on the TAS to use certain OS libraries or other components that may not be available in all the test environments of the SUT

Typical deployment project risks include:

- Staffing issues: Getting the right people to maintain the test automation may be difficult
- Unplanned TAS maintenance due to SUT updates that cause the TAS to operate incorrectly
- Delays in introducing test automation
- Delays in updating the TAS based on the changes done to the SUT
- The TAS cannot capture nonstandard UI objects
- Allowing outdated test cases to remain in test suites, wasting test execution time

Potential failure points of the TAS project include:

- Migration to a different test environment
- Deployment to a production environment
- Forgetting that automation is software that should also be tested

It is important to realize that various technical issues, project risks, and potential failure points can jeopardize the success of test automation projects. Common technical issues include complexities arising from excessive abstractions, challenges with test data management, and dependencies on specific components. Project risks also need to account for difficulties in staffing, maintenance issues due to system updates, deployment delays, and the presence of outdated test cases. Additionally, potential failure points such as migration to different environments and neglecting the need to test the automation software itself, should be addressed with proper planning. Overall, monitoring and proactive measures will ensure the success of test automation projects.

4.2.3 Define Approaches To Mitigate Deployment Risks

There are many risk mitigation strategies that can be employed to deal with these risk areas. Some of these are discussed below.

The TAS has an SDLC of its own, whether it is in-house developed or an acquired solution. One thing to remember is that the TAS, like any other software, needs to be under configuration management and its features documented. Otherwise, it becomes exceedingly difficult to deploy different parts of it and make them work together or work in certain test environments.

Also, there has to be a documented, clear, and easy to follow deployment procedure. This procedure is version dependent; therefore, it has to be included under configuration management as well.

There are two distinct cases when deploying a TAS:

1. First-time deployment
2. Maintenance deployment – the TAS already exists, and an update needs to be deployed

The risks related to the first-time deployment include:

- Total test execution time of the test suite may be longer than the planned test execution time for the test cycle. In this case it is important to make sure to plan enough time for the test suite to execute in its entirety before the next scheduled test cycle begins.
- Installation and configuration issues with test environments exist (e.g., database setup and initial load, and services start/stop). The TAS needs a test fixture (i.e., a predefined dataset) to create the necessary preconditions for automated test cases to run within the test environment.

For maintenance deployments, there are additional considerations. The TAS in itself needs to evolve, and the updates for it have to be deployed into production. Before deploying an updated version of the TAS into production, it needs to be tested. It is therefore necessary to check the new functionality, to verify that a test suite can be run on the updated TAS, that test reports can be sent, and that there are no performance defects or other quality issues. In some cases, the entire test suite may need to be changed to fit the latest version of the TAS.

4.3 Dependencies within the Test Environment

4.3.1 Define test automation components in the test environment

Test automation components typically consist of tools, virtual machines, automation test scripts, containers, and configurations. The test environment also includes the SUT. The test automation components in the test environment can be defined as follows:

SUT – This is an obvious part of the test environment. The SUT can be tested as a whole or divided into subcomponents (e.g., API, Web-based interface, and database).

Platform – The platform describes where the test automation components are hosted. This includes the cloud infrastructure, network, virtual machines, and containers that may be used to make test automation efficient and portable.

Test cases and test suites – This describes individual test cases that are made up of test steps that direct both automated and manual actions. The test suites describe a logical grouping of test cases so that they can be executed together efficiently.

Tools – Different test automation tools are used for several reasons. A collection of tools would include those for UI test automation, testing API endpoints, data generation, monitoring, requirements management, defect management, logging and reporting tools, and tools for generating trends based on metrics.

TAF – This includes all the items that go into making up the TAF design. TAFs include driver scripts, common libraries, templates for automated test cases, data loading/provisioning scripts, documentation on how to use TAF components, and tutorials that help TAEs utilize the TAF. For additional information, see the CTAL-TAE Syllabus, section 3.1.

Maintenance of the test components is a major consideration because overcomplicating the test environment may result in hours of unwanted time fixing defects in the TAF instead of benefiting from the solution. It is important to find the right blend of tools, configuration, and platform portability to make the components as reusable as possible.

4.3.2 Identify Infrastructure Components and Dependencies of Test Automation

There are a number of infrastructure components and dependencies to be aware of when assembling test automation. Collectively they cover all of the prerequisites necessary to run a TAS. The major components and dependencies include:

Host machines – These can be virtual machines, physical servers, laptops, and devices (e.g., tablet, and mobile). They have the test automation software installed on them and that is where test scripts are created and executed.

Network – This is what gives the TAS access to the SUT. It can also include the networking together of multiple host machines to provide parallel execution of automated tests. Typically, host machines are required to be on the same network and configured properly to communicate with each other.

Platform – Test automation, like any other software, can run on cloud platforms or designed to run in containers. As long as the platform provides permissions and access to the underlying OS, all necessary tools and dependencies can be installed.

Software dependencies – To make the test automation tools perform correctly, it is necessary to understand all other dependencies that the test automation has. For example, a particular test automation tool may require the latest version of a programming language to be installed on the host machine first. TAEs need to account for these dependencies before and after selecting a tool.

SUT – Once the components, dependencies, and overall infrastructure have been considered and configured properly, the last step is to ensure access to the SUT. In addition to the network, it is also necessary to consider how to interface with the SUT. For example, in a Web-based application, a browser needs to be installed on the host machine. The type of browser and version needs to be considered.

4.3.3 Define Test Automation Data and Interface Requirements

Two considerations TAEs need to think about before developing test automation scripts are how they plan to interface with the SUT and what are the data dependencies of the test cases. The following are examples:

API – If the SUT utilizes an API, this can be tested at the end point level instead of requiring a UI to interface at the application level. This may require a deeper understanding of the end points and data communications that are hidden by a UI. The TAE would have to understand in what order to call APIs to create a business process and how to correlate the data to keep the test case intact. APIs that are exposed to the Internet are also called web APIs or web services.

Database interface – Some test automation tools have the ability to interface directly with the SUT's underlying database. Test scripts can be written to verify data within columns and rows to ensure that stored procedures and other database rules are configured properly. This may require specific data values to already exist in the database for reliability testing.

Interface compatibility – Use of contract testing ensures that two separate systems (e.g., two microservices) are compatible and are able to communicate with one another. Contracts testing can be consumer-driven, producer-driven, and bidirectional. Additional details can be found in the CTAL-TAE Syllabus, section 5.1.3.

Furthermore, TAEs must carefully consider how they will interface with the SUT and understand the data dependencies within test cases. Whether testing through APIs, database interfaces, or ensuring interface compatibility between systems, attention to these considerations is essential for robust and effective test automation. By addressing these factors thoughtfully, TAEs can enhance the reliability and efficiency of their automation efforts, ultimately contributing to the quality and success of the SUT.

5 Test Automation Impact Analysis – 150 minutes (K3)

Keywords

coverage, test report

Learning Objectives for Chapter 5:

5.1 Investment in Setting Up and Maintaining Test Automation

CT-TAS-5.1.1 (K3) Show the return on investment of building a test automation solution

5.2 Test Automation Metrics

CT-TAS-5.2.1 (K2) Classify metrics for test automation

5.3 The Value of Test Automation on the Project and Organization Level

CT-TAS-5.3.1 (K3) Identify organizational considerations for use of test automation

CT-TAS-5.3.2 (K3) Analyze project characteristics that help determine optimal implementation of test automation test objectives

5.4 Decisions Made from Test Automation Reports

CT-TAS-5.4.1 (K2) Analyze test report data to inform decision making

5.1 Investment in Setting Up and Maintaining Test Automation

5.1.1 Show Return On Investment of Building a Test Automation Solution

It is important to estimate the setup and maintenance costs of test automation before proceeding to start a project implementation. Beyond the values that automation can bring to a project, understanding the ROI calculation for the project is beneficial.

The ROI calculation can provide meaningful feedback at any given moment in a project lifecycle by demonstrating the return for an activity for the effort invested. The ROI calculation can be further adjusted by including both manual tester costs and TAE costs by simply multiplying the time spent with their respective labor costs.

To calculate the ROI, one needs to determine the investment of test automation (i.e., time and cost) and the savings achieved with it:

ROI = Savings / Investment

It is important to note that the savings and the investment can be calculated considering different metrics and data, in different measures and units. In the scope of this syllabus, a simple model is used to demonstrate the approach, with time units and not the cost. If certain activities are only measured in cost, that amount can be converted to time using a rate specific to the project.

Generally, the savings achieved by test automation are because the same tests can be run in significantly shorter time than run manually. This also means they can be run more often. Therefore, the number of tests being executed can be increased.

To calculate savings, you need to consider the following metrics:

- Time to run a test case manually
- Time to run an automated test case
- Number of test cases
- Number of test runs

To calculate investment, you need to consider the following metrics:

- Time to set up test automation
- Average time to develop automated test scripts
- Number of automated test scripts implemented
- Average maintenance time of an automated test script
- Time to run an automated test script
- Percentage of failed automated test scripts
- Number of test cases defined
- Number of test runs

By adapting the outlined model to Agile software development, a project's sprints can be forecast as illustrated in the graph below. One can determine the sprint from which test automation has returned its investment by using the graph.

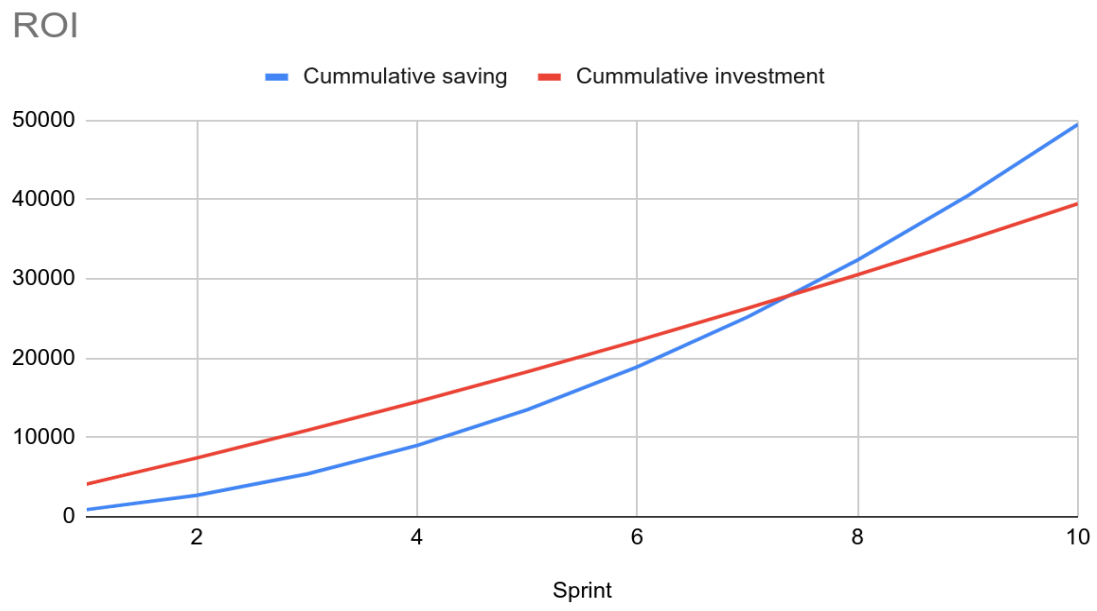

Figure 2: An Example of an ROI calculation showing the point of Return on Investment

Keep in mind certain connections between the metrics measured to calculate the ROI, including:

- If the planned project duration is less than the turning point of the ROI, it is not worth introducing test automation. In this case, by executing the tests manually, time and effort are saved.
- As the investment depends significantly on the execution time of the automated tests, applying the test pyramid and implementing test cases at the right test level can reduce this execution time and improve ROI.

5.2 Test Automation Metrics

5.2.1 Classify Metrics for Test Automation

Analyzing trends based on factual data helps in decision making. By collecting metrics of a TAS, the testers are able to evaluate the TAS and make decisions on:

- TAS suitability to the project
- TAS adaptability for expanded functionality for new test conditions
    - A change in a user flow in the SUT
    - A change in how testing is performed
- Maintainability of test automation because of defects found in the TAS

The cost of measuring should be as low as possible, and this can often be achieved by automating the collection and reporting of metrics. Some examples can be seen below.

Pass-fail ratio

This is a common metric and tracks the ratio of the automated tests that passed to the automated tests that failed to achieve the expected result. The pass-fail ratio = the number of tests that passed / the number of tests that failed.

Ratio of failures to defects

A common problem with automated tests is that many of them can fail for the same reason, i.e., a single defect in the SUT. Measuring the number of automated tests that fail for a given defect can help indicate where this may be a problem. Additionally, one may collect the number of failures per defect and track the reason for failure: defect in the sub-system, in the whole system, issue with the test data, or infrastructure.

Test automation execution time

One of the easier metrics to determine is the time it takes to execute the automated tests. In the beginning of the TAS this might not be important, but as the number of automated test cases increases, this metric may become quite important. This metric includes the build time of the TAS.

Number of automated test cases

This metric can be used to show the progress made by the test automation project. But one has to take into account that the number of automated test cases does not reveal much information; for example, it does not indicate the level of coverage.

Functional coverage of test automation

This coverage indicates the percentage of functional requirements covered by automated test cases.

Code coverage

Code coverage tracks how many lines of code is exercised by lower level (i.e., component) tests.

There is no absolute percentage that indicates adequate coverage, and 100% code coverage is often difficult to attain in anything other than the simplest software. However, it is generally agreed that more coverage is better as it increases confidence in the SUT. As additional low-level automated tests are added, code coverage is expected to increase.

5.3 The Value of Test Automation on the Project and Organization Level

5.3.1 Identify Organizational Considerations for Use of Test Automation

Before starting test automation on any project, one needs to identify the following within the organization:

Policies and practices for software development

It is good to check how the development teams are working and what kind of documentation is present. Any available documentation can be beneficial to identify how test automation can be connected into the processes of the development teams. Documentation can include the SUT's technical specification, the software and development tools used, or any available policies about development, such as code review guidelines, defined coding standards, and merge processes.

Existing active test automation projects and their status

In the case of an ongoing development project, there can be one or more ongoing TAS development projects by different teams. A general recommendation for the decision makers is to check these existing projects and their status, to see if any of them fit within the defined test objectives of the new TAS. By analyzing the solution, one can determine whether to reuse any of the existing TAS or to create a new one based on the current needs.

The organization's subject matter experts for test automation

It is recommended to look for SMEs who can assist in the roll out of a new TAS within the organization. The SMEs can share stories and lessons learned about their TAS and their rollout. From this, one can identify different risks that need to be considered or avoided to have a successful TAS rollout.

Availability of test environments

In the case of large organizations, a general recommendation is to gather information about the existing test environments, their use, and availability. This information is crucial to avoid ruining any other team's work by rolling out a new TAS and using the system without any notification or agreement. Often, the needs of a project will require a new test environment, so the existing ones can be used as a baseline in creating a new one and the work can be done more easily if the responsible teams and persons are identified and available.

Test tools and licenses

It is always worthwhile to get information about what tools and licenses the organization currently has. By identifying these, costs, and timelines, planning of a TAS can be reduced. And tools for a new project may already be available. For example, if cloud testing is set up through a defined provider, and the licenses are available for the new project, it does not make sense to use a different cloud provider. It is recommended to use the same tools and licenses to reduce project costs.

5.3.2 Analyze Project Characteristics that Help Determine Optimal Implementation of Test Automation Test Objectives

There are several major project characteristics that can define an optimal way of working, and to help define successful test automation objectives.

Domain

It is important to understand that each domain differs either in regulations or standards. For example, there are different regulations and risks in the tourism domain compared to finance or healthcare. It is always recommended to check the different standards and domain restrictions to make sure that the planned test automation objectives comply.

Platforms

In terms of test automation test objectives, it is also emphasized to evaluate which platforms the project covers and where it would be beneficial to do test automation. Planning for multiple platform test automation can be more difficult since there may be a need for multiple solutions to cover the different platforms. In many cases, such as for mobile and web, the same tools can be used. But planning will determine reusability of the TAS.

Programming language and technology stack

The implementation details of a project, like its programming language and technology stack, also determine what kind of test automation test objectives an organization should define. It is recommended to use the same programming languages that the developers are using on a given project. By doing so, collaboration can be much easier, and cross learning by carrying out code reviews together will improve the quality of the SUT and the knowledge of the TAEs even further.

Maturity of the project

By analyzing the maturity of the project, information can be derived to design the ideal test automation objective. By identifying the different factors such as the number of test cases, existing CI/CD pipelines, number of testers and TAEs, different test objectives and a roadmap can be created. For example, if there are a lot of manual test cases that have high priority and their manual test execution time is long, it is important to aim to get these automated first to save time and effort. Apart from the above factors, the timeline of the project is also very important. If the project is short or will end shortly and there is not much time to get things done, it is not worth planning a big and robust TAS. However, if the project is a greenfield project, it is worth devising a step-by-step minimum viable product-based incremental roadmap.

Stakeholder buy-in

Many times, test automation is not leveraged due to key stakeholders not buying into it. That can be due to fears of velocity dropping below their target, tight deadlines, budgeting, or other concerns. After analysis of the project maturity, if there is room to start implementing a TAS, a strategic stakeholder needs to identify the product risks, list the benefits of introducing test automation, and come up with an appropriate plan. This plan needs to highlight the milestones to be achieved with and by test automation and indicate how the testability of the SUT needs to be improved to successfully introduce a TAS that will address the identified risks. Lastly, this plan needs to be presented to the stakeholders.

Team knowledge and relevant experience

The most important and relevant factor for successful test automation is a testers' skills and experience. It is beneficial to work with the testers and use their knowledge to define test objectives that they and the business analysts are comfortable with. Often test managers or test leaders create a competency and skill matrix to identify the available knowledge within the teams, and to identify the gaps at the same time. These gaps can be identified with different goals such as training and assigned implementation tasks.

Test management support and budgeting

Depending on the size and maturity of the given project, the available budget and test management support should be considered, while defining the test objectives for test automation. It is important to define test objectives that can be met, which in return will enable support from test management.

When making a test automation strategy proposal to management, the documentation needs to be concise, clearly defining currently identified gaps or future costs for test automation is implemented correctly. A list of recommendations and their benefits pointing out the business value and cost reductions will encourage test management to approve developing or improving a TAS.

Quality characteristics

The ISO/IEC 25010:2011 defines the quality characteristics that are listed in the ISTQB CTFL Syllabus, section 2.2.2. Test Types. These quality characteristics can then be used for evaluating the current TAS, or determining metrics that will be collected by the TAS.

5.4 Decisions Made from Test Automation Reports

5.4.1 Analyze Test Report Data to Inform Decision Making

The format and the content of a test automation report may vary depending on the stakeholders receiving it. It can be created for management, operational or technical stakeholders. Additional information can be found in the CTAL-TAE Syllabus, section 6.1.3.

Upon receiving a test automation report, it can be incorporated into a broader test report, or it can be consolidated, and then escalated within the organizational structure.

Different stakeholders find different values in such a test automation report. A strategic approach is to identify key metrics that are important for the stakeholders involved by emphasizing these important metrics.

Data collected with automation can help:

- Identify trends and perform root cause analysis
- Shift test automation efforts to maintenance
- Shift test automation efforts to improvements and further development of a TAF
- Add capabilities to the overall TAS
- Increase shift left and shift right approaches to testing
- Expand the functional coverage of test automation in future sprints
- Focus more on defect clusters
- Advise developers of areas to improve the code
- Advise on the overall SDLC processes
- Change the content and format of future test automation reports

Based on the information described above, the TAEs in collaboration with other stakeholders can identify gaps and certain improvement points in the existing test automation coverage and test results.

6 Implementation and Improvement Strategies for Testing Automation – 150 minutes (K3)

Keywords

coverage, precondition, test suite

Learning Objectives for Chapter 6:

6.1 Transitioning Activities from Manual Testing to Continuous Testing

CT-TAS-6.1.1 (K2) Describe the factors and planning activities in transitioning from manual testing to test automation

CT-TAS-6.1.2 (K2) Describe the factors and planning activities in transitioning from test automation to continuous testing

6.2 Understanding the Factors and Planning Activities in Transitioning from Test Automation to Continuous Testing

CT-TAS-6.2.1 (K3) Conduct an evaluation of the test automation assets and practices to identify improvement areas

6.1 Transitioning Activities from Manual Testing to Continuous Testing

6.1.1 Describe the Factors and Planning Activities in Transitioning from Manual Testing to Test Automation

The easiest opportunity for transitioning from manual testing to test automation is to target regression testing. A regression test suite grows as today's functional and non-functional tests become tomorrow's regression tests. It is only a matter of time before the number of regression tests becomes greater than the time and resources available to a traditional manual test team.

Transitioning costs

During transition from manual testing to automated testing, the project should expect increased costs as both manual testing and automated testing take place simultaneously. Once automated tests are deemed to adequately replace or supersede manual test execution activities, more effort can be shifted towards exploratory testing and defining additional test cases for test automation, which has a different cost compared to manual regression testing.

Functional overlap

Functional overlap occurs when test script developers include the same exact test automation steps in different test cases. For example, most test cases will start with a login sequence of test steps. This can include entering a username, password, and selecting a login button. Adding it in each test case increases maintenance activities. If there is ever an additional test step added to the login process, the same change will have to be updated in every test case. A better approach is to make a repeatable test automation component out of the login process and have all test cases reference that functionality. See the CTAL-TAE Syllabus, section 3.1.5 for details about the flow model pattern.

Data sharing

Tests often share test data. This can occur when tests use the same record of test data to execute different SUT functionality. An example of this might be test case "A" which verifies an employee's available vacation time, while test case "B" verifies what courses the employee took as part of their career development goals. Each test case uses the same employee but verifies different parameters. In a manual test environment, the employee test data would typically be duplicated many times across each manual test case which verified employee test data using this employee. However, in an automated test, test data which is shared should, where possible and feasible, be stored and accessed from a single source to avoid duplication, or the introduction of errors.

Test interdependency

When executing complex regression tests, one test may have a dependency on one or more other tests. This occurrence can be quite common. For example, a new "Order ID" gets created as a result of a test step. Subsequent tests may want to verify that: a) the new order is correctly displayed in the system, b) changes to the order are possible, or c) deleting the order is successful. In each case, the "Order ID" value which is dynamically created in the first test must be captured for reuse by later tests. Depending on the design of the TAS, this can be addressed. If the "Order ID" cannot be found by subsequent test cases, they will fail.

Test execution preconditions

All too often TAEs will immediately begin test script development without first understanding the preconditions that go into making sure a test case can execute reliably. This can become extremely challenging when trying to execute a test case against the same SUT in multiple test environments. Examples of preconditions include usernames, account roles, data table values, and other specific data entries that make the test case repeatable. A good strategy will include upfront analysis to understand what information needs to exist in the SUT before automating a particular test case. Additionally, the strategy can be enhanced by automating the creation of preconditions prior to the real test cases running, to save time. This may include running precondition test scripts that populate the SUT from the UI or automated batch processes that load test data to a database.

Functional coverage

Identify the functional gaps in testing that can be candidates for test automation, as explained in Chapter 3. 100% of the manual test cases that are automated does not represent 100% of all possible test cases that can be automated with test tools. As more tests become automated, testers get back test execution time which they can use to identify additional SUT tests to increase coverage.

Executable tests

Before converting a manual regression test into an automated regression test, it is important to verify that the manual regression test operates correctly. This then provides the correct starting point to ensure a successful conversion to an automated regression test. If the manual regression test does not execute correctly, it may be because it was poorly written, uses invalid test data, is out of date or out of sync with the current SUT, or because of an SUT defect. Automating it prior to understanding and/or resolving the root cause of the failure will create a non-functioning automated regression test which is wasteful and unproductive. It is important to demonstrate equivalent functionality that the new automated tests bring, to convey confidence in automated tests that will replace the old manual tests.

6.1.2 Describe the Factors and Planning Activities in Transitioning from Test Automation to Continuous Testing

Continuous testing involves utilizing test assets far more frequently than in traditional SDLCs. This is done by starting to constantly run test suites immediately after code changes are done and made available in the test environments. This helps provide immediate feedback and reduces defect costs by catching defects earlier in the process. This can be handled with sophisticated development tools and the willingness to shift testing earlier in the build process.

Adapting the TAS for continuous testing requires the following:

- The test suites need to change to run on the updated TAS: make the necessary changes to the test suites and test them before deploying them on the TAS
- Stubs, drivers, and interfaces used in testing need to change to fit with the updated TAS: make the necessary changes to the test harness and test it before deploying it on the TAS
- The infrastructure needs to change to accommodate the updated TAS: make an assessment of the infrastructure components that need to be changed
- The updated TAS has additional defects or performance defects: perform an analysis of risks versus benefits. If the defects discovered make it impossible to update the TAS, it may be best not to proceed with the update or to wait for a next version of the TAS. If the defects are negligible compared to the benefits of correcting them, the TAS can still be updated.
- Be sure to create release notes of known defects to notify the TAEs and other stakeholders and try to get an estimate of when the defects are going to be fixed

All of the points mentioned in this section become particularly important when utilizing CI/CD. Putting together pipelines and automating the build process is a natural fit with the TAS being developed. If the build orchestration tool is in the correct test environment, the pipeline can be extended to include automated tests to verify the SUT right after deployment. This requires that the test automation tool be properly configured and can access the SUT, and that it can reach out to the code repository and access data provisioning scripts.

6.2 Test Automation Strategy Across the Organization

6.2.1 Conduct an Evaluation of the Test Automation Assets and Practices to Identify Improvement Areas

Just like any other development activity, it is important to have a strategy for pausing the TAS development and looking for opportunities to refactor the solution. The areas to monitor are initial implementation, maintenance, and the ability to evaluate the solution from a repeatability standpoint. Some good categories to keep track of are how many hours are spent developing the TAS, how many hours are spent fixing the TAS, and how much time savings do the testers realize compared to manual testing. An indicator that something is wrong is if the TAS maintenance time is greater than the manual testing time.

Before starting with the first deployment of a TAS, it is important to be sure it can run in its own environment, it is isolated from random changes, and test cases can be updated and managed. Both the TAS and its infrastructure must be maintained. In the case of first-time deployment, the following basic steps are needed:

- Code coverage tools indicate how much of the code is exercised by the component test suite, and where gaps exist that can be covered through additional component tests
- SUT functional coverage can be determined by creating a requirements traceability matrix, which uncovers which functionality is not yet covered by any test cases
- Define a consistent infrastructure usage of the TAS across projects or the whole organization
- Develop a consistent configuration management strategy for the test suites
- Create common TAS development guidelines leveraging best practices described in the CTAL-TAE Syllabus, section 3.1.4
- Implement preconditions. Often a test cannot be executed prior to setting preconditions. These preconditions may include selecting the correct database or the test data from which to test or setting initial values or parameters. Many of these initialization steps that are required to establish a test's preconditions can be automated. This allows for a more reliable and dependable solution when these steps cannot be missed prior to executing the tests. As regression tests are converted to test automation, these preconditions need to be a part of the test automation process.

When incremental TAS updates occur for new features or maintenance purposes, the following should be considered:

- Evaluate updates to test tools or other newer test tools that can provide extended capabilities for the TAS
- Evaluate ways in which to further optimize TAS features and performance
- Identify opportunities to further decompose and modularize test scripts to enhance reusability
- Ensure knowledge and awareness of reusable components and consistent utilization
- Collect evidence on potential improvement areas and to provide recommendations and their benefits
- Evaluate and correct areas of functional overlap. When automating existing regression tests, it is good practice to identify any functional overlap that exists between test cases and, where possible, reuse previously developed test automation components.
- Evaluate additional manual test cases for opportunity to automate them and create backlog items for implementation
- Highlight test design and test data management improvement opportunities
- Ensure all existing test suites are adapted to the latest version of the TAS
- If test automation is causing the pipeline to queue, decrease the scope of integrated tests to the most critical ones, i.e., create a smoke test suite. The larger regression test suite can be triggered separately or executed on demand.

7 References

Standards

Standards for test automation include but are not limited to:

The Automatic Test Markup Language (ATML) by IEEE (Institute of Electrical and Electronics Engineers) consisting of: • IEEE Std 1671.1: Test Description • IEEE Std 1671.2: Instrument Description • IEEE Std 1671.3: UUT Description • IEEE Std 1671.4: Test Configuration Description • IEEE Std 1671.5: Test Adaptor Description • IEEE Std 1671.6: Test Station Description • IEEE Std 1641: Signal and Test Definition • IEEE Std 1636.1: Test Results
ISO/IEC 30130:2016 (E) Software engineering — Capabilities of software testing tools
The Testing and Test Control Notation (TTCN-3) by ETSI (European Telecommunication Standards Institute) and ITU (International Telecommunication Union) consisting of: • ES 201 873-1: TTCN-3 Core Language • ES 201 873-2: TTCN-3 Tabular Presentation Format (TFT) • ES 201 873-3: TTCN-3 Graphical Presentation Format (GFT) • ES 201 873-4: TTCN-3 Operational Semantics • ES 201 873-5: TTCN-3 Runtime Interface (TRI) • ES 201 873-6: TTCN-3 Control Interface (TCI) • ES 201 873-7: Using ASN.1 with TTCN-3 • ES 201 873-8: Using IDL with TTCN-3 • ES 201 873-9: Using XML with TTCN-3 • ES 201 873-10: TTCN-3 Documentation • ES 202 781: Extensions: Configuration and Deployment Support • ES 202 782: Extensions: TTCN-3 Performance and Real-Time Testing • ES 202 784: Extensions: Advanced Parameterization • ES 202 785: Extensions: Behaviour Types • ES 202 786: Extensions: Support of interfaces with continuous signals • ES 202 789: Extensions: Extended TRI
The UML Testing Profile (UTP) by OMG (Object Management Group) specifying test specification concepts for: • Test Architecture • Test Data • Test Behavior • Test Logging • Test Management

ISTQB® Documents

Identifier	Reference
ISTQB-AL-TM	ISTQB Certified Tester, Advanced Level Syllabus, Test Manager, Version 2.0, October 2012, available from [ISTQB-Web]
ISTQB-EL-TM-MTT	ISTQB Certified Tester, Expert Level Test Management Managing the Test Team, Version 2.0, November 2011, available from [ISTQB-Web]
ISTQB-FL	ISTQB Certified Tester, Foundation Level Syllabus, Version 4.0, April 2023, available from [ISTQB-Web]
ISTQB-PT	ISTQB Certified Tester, Performance Testing Syllabus, December 2018, available from [ISTQB-Web]
ISTQB-TAE	ISTQB Certified Tester, Test Automation Engineering Syllabus, February 2024, available from [ISTQB-Web]
ISTQB-Glossary	ISTQB Glossary of terms, available online from [ISTQB-Web]

Books

Paul Baker, Zhen Ru Dai, Jens Grabowski and Ina Schieferdecker, “Model-Driven Testing: Using the UML Testing Profile”, Springer 2008 edition, ISBN-10: 3540725628, ISBN-13: 978-3540725626
Efriede Dustin, Thom Garrett, Bernie Gauf, “Implementing Automated Software Testing: how to save time and lower costs while raising quality”, Addison-Wesley, 2009, ISBN 0-321-58051-6
Efriede Dustin, Jeff Rashka, John Paul, “Automated Software Testing: introduction, management, and performance”, Addison-Wesley, 1999, ISBN-10: 0201432870, ISBN-13: 9780201432879
Mark Fewster, Dorothy Graham, “Experiences of Test Automation: Case Studies of Software Test Automation”, Addison-Wesley, 2012
Mark Fewster, Dorothy Graham, “Software Test Automation: Effective use of test execution tools”, ACM Press Books, 1999, ISBN-10: 0201331403, ISBN-13: 9780201331400
Boby Jose, “Test Automation, a manager's guide”, September 2021, ISBN: 9781780175478
James D. McCaffrey, “.NET Test Automation Recipes: A Problem-Solution Approach”, APRESS, 2006 ISBN-13:978-1-59059-663-3, ISBN-10:1-59059-663-3
Daniel J. Mosley, Bruce A. Posey, “Just Enough Software Test Automation”, Prentice Hall, 2002, ISBN-10: 0130084689, ISBN-13: 9780130084682
Casey Rosenthal, “Chaos Engineering: System Resiliency in Practice by Casey Rosenthal”, April 2020, ISBN-13: 1492043869
Colin Willcock, Thomas Deiß, Stephan Tobies and Stefan Keil, “An Introduction to TTCN-3” Wiley, 2nd edition 2011, ISBN-10: 0470663065, ISBN-13: 978-0470663066

Articles

Robert V. Binder, Suzanne Miller, “Five Keys to Effective Agile Test Automation for Government Programs” August 24, 2017, Software Engineering Institute, Carnegie Mellon University, https://resources.sei.cmu.edu/asset_files/Webinar/2017_018_101_503516.pdf
DoD CIO, Modern Software Practices “DevSecOps Fundamentals Guidebook: Activities & Tools”, Version 2.2, May 2023, https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsActivitesToolsGuidebookTables.pdf?ver=_Sylg1WJB9K0Jxb2XTvzDQ%3d%3d
Naveen Jayachandran, “Understanding roi metrics for software test automation”, 2005, https://digitalcommons.usf.edu/cgi/viewcontent.cgi?article=3937&context=etd
Thomas Pestak, William Rowell, PhD, “Automated Software Testing Practices and Pitfalls”, September 2018, https://www.afit.edu/stat/statcoe_files/Automated%20Software%20Testing%20Practices%20and%20Pitfalls%20Rev%201.pdf
Andrew Pollner, Jim Simpson, Jim Wisnowski, “Automated Software Testing Implementation Guide for Managers and Practitioners”, October 2018, https://www.afit.edu/stat/statcoe_files/0214simp%202%20AST%20IG%20for%20Managers%20and%20Practitioners.pdf

8 Appendix A – Learning Objectives/Cognitive Level of Knowledge

The following learning objectives are defined as applying to this syllabus. Each topic in the syllabus will be examined according to the learning objective for it.

The learning objectives begin with an action verb corresponding to its cognitive level of knowledge as listed below.

Level 2: Understand (K2)

The candidate can select the reasons or explanations for statements related to the topic, and can summarize, compare, classify, and give examples for the testing concept.

Action verbs: Classify, compare, differentiate, distinguish, explain, give examples, interpret, summarize

Examples	Notes
Classify test tools according to their purpose and the test activities they support.	
Compare the different test levels.	Can be used to look for similarities, differences, or both.
Differentiate testing from debugging.	Looks for differences between concepts.
Distinguish between project and product risks.	Allows two (or more) concepts to be separately classified.
Explain the impact of context on the test process.	
Give examples of why testing is necessary.	
Infer the root cause of defects from a given profile of failures.	
Summarize the activities of the work product review process.	

Level 3: Apply (K3)

The candidate can carry out a procedure when confronted with a familiar task or select the correct procedure and apply it to a given context.

Action verbs: Apply, implement, prepare, use

Examples	Notes
Apply boundary value analysis to derive test cases from given requirements.	Should refer to a procedure / technique / process etc.
Implement metrics collection methods to support technical and management requirements.	
Prepare test automation environment.	
Use traceability to monitor test progress for completeness and consistency with the test objectives, test strategy, and test plan.	Could be used in a LO that wants the candidate to be able to use a technique or procedure. Similar to 'apply'.

Reference

(For the cognitive levels of learning objectives)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon

9 Appendix *B* – Business Outcomes traceability matrix with Learning Objectives

This section lists the traceability between Test Automation Strategy Specialist Business Outcomes and Test Automation Strategy Specialist Learning Objectives.

Business Outcomes Test Automation Strategy Specialist			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12	B13	B14	B15
Chapter 1	Introduction and Objectives for Test Automation Strategy																
1.1	Success Factors of a Test Automation Project																
1.1.1	Explain the Objectives and Relevance of Test Automation	2	X														
1.1.2	Identify Technical Success Factors of a Test Automation Project	2	X														
1.1.3	Summarize Appropriate Investment Criteria in Selecting Candidate Projects for Test Automation	2	X														
Chapter 2	Test Automation Resources																
2.1	Costs and Risks of Implementing a Test Automation Solution																
2.1.1	Compare Alternative Technical Solutions with Regard to Cost of Ownership	2		X													
2.1.2	Explain Licensing Model Considerations for Test Automation Tools	2		X													
2.1.3	Provide Examples of Factors to be Considered When Defining a Test Automation Strategy	2		X													
2.2	Roles and Responsibilities within Test Automation																
2.2.1	Summarize the Roles and Skills Necessary for a Successful Test Automation Solution	2			X												

Business Outcomes Test Automation Strategy Specialist			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12	B13	B14	B15
Chapter 3	Preparing for Test Automation																
3.1	Integration Across Test Levels																
3.1.1	Differentiate Between Test Automation Distributions	2				X											
3.1.2	Select a Test Automation Approach Based on the System Under Test Architecture	3				X											
3.1.3	Demonstrate Ways to Optimize Test Automation Distribution to Achieve Shift Left and Shift Right Approaches	2				X											
3.2	Strategic Considerations in Different Software Development Lifecycle Models																
3.2.1	Explain how test automation projects conform with legacy software development lifecycle models	2					X										
3.2.2	Explain How Test Automation Projects Conform with Agile Software Development Best Practices that Support Test Automation	2					X										
3.2.3	Prepare for Test Automation Projects to Conform with DevOps Best Practices that Support Test Automation in Continuous Testing	3					X										
3.3	Applicability and Viability of Test Automation																
3.3.1	Explain Criteria for Determining the Suitability of Tests for Test Automation	2						X									
3.3.2	Identify Challenges that Only Test Automation can Address	2						X									
3.3.3	Identify Test Conditions that are Difficult to Automate	2						X									

Business Outcomes Test Automation Strategy Specialist			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12	B13	B14	B15
Chapter 4	Organizational Deployment and Release Strategies for Test Automation																
4.1	Test Automation Solution Planning																
4.1.1	Identify ways how Test Automation supports shorter time to market	2							X								
4.1.2	Identify Ways in Which Test Automation Helps Verify Reported Defects According to Requirements	2							X								
4.1.3	Define Approaches that Allow for the Development of Operationally Relevant Scenarios for Test Automation	2							X								
4.2	Deployment Strategies																
4.2.1	Define a Test Automation Deployment Strategy	2								X							
4.2.2	Identify Test Automation Risks in Deployment	2								X							
4.2.3	Define Approaches To Mitigate Deployment Risks	2								X							
4.3	Dependencies within the Test Environment																
4.3.1	Define Test Automation Components in the Test Environment	2									X						
4.3.2	Identify Infrastructure Components and Dependencies of Test Automation	2									X						
4.3.3	Define Test Automation Data and Interface Requirements for Integration within the System Under Test	2									X						
Chapter 5	Test Automation Impact Analysis																
5.1	Investment in Setting Up and Maintaining Test Automation																

Business Outcomes Test Automation Strategy Specialist			B01	B02	B03	B04	B05	B06	B07	B08	B09	B10	B11	B12	B13	B14	B15
5.1.1	Show Return On Investment of Building a Test Automation Solution	3										X					
5.2	Test Automation Metrics																
5.2.1	Classify Metrics for Test Automation	2											X				
5.3	The Value of Test Automation on the Project and Organization Level																
5.3.1	Identify Organizational Considerations for Use of Test Automation	3												X			
5.3.2	Analyze Project Characteristics that Help Determine Optimal Implementation of Test Automation Test Objectives	3												X			
5.4	Decisions Made from Test Automation Reports																
5.4.1	Analyze Test Report Data to Inform Decision Making	2													X		
Chapter 6	Implementation and Improvement Strategies for Test Automation																
6.1	Transitioning Activities from Manual Testing to Continuous Testing																
6.1.1	Describe the Factors and Planning Activities in Transitioning from Manual Testing to Test Automation	2														X	
6.1.2	Describe the Factors and Planning Activities in Transitioning from Test Automation to Continuous Testing	2														X	
6.2	Test Automation Strategy Across the Organization																
6.2.1	Conduct an Evaluation of the Test Automation Assets and Practices to Identify Improvement Areas	3															X

10 Appendix C – Release Notes

ISTQB® Test Automation Strategy Syllabus 2024 is a new ISTQB syllabus that combines strategic aspects from the prior ISTQB Test Automation Engineer syllabus release from 2016 with additional updates and current best practices for implementing and measuring success of test automation. For this reason, there are no detailed release notes per chapter and section.

11 Appendix D – Domain-Specific Terms

Term Name	Definition
canary release	A deployment and testing strategy meant to reduce risk and verify new software by releasing software to only a few users.
container	A unit of software that packages code and its dependencies, so the software runs quickly and reliably across environments.
DevOps	A methodology that integrates and automates the work of software development and IT operations to improve and shorten the software development lifecycle.
flow model pattern	A high-level view of the work domain, its components, and interconnections among them.

12 Index

All terms are defined in the ISTQB® Glossary (http://glossary.istqb.org/).