SYLLABUS

Version 2022

AT*SQA
ASSOCIATION FOR TESTING & SOFTWARE QUALITY ASSURANCE
Global Certification Body for ISTQB and ASTQB

0. Introduction to this Syllabus

0.1. Purpose of this Document

This syllabus forms the basis of the AT*SQA certification for Testing Essentials. AT*SQA is an International Standards Organization (ISO) compliant certification body for software testers. AT*SQA provides this syllabus as follows:

1. To training providers - to produce courseware and determine appropriate teaching methods.
2. To certification candidates - to prepare for the exam (as part of a training course or independently).
3. 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.

AT*SQA may allow other entities to use this syllabus for other purposes, provided they seek and obtain prior written permission.

0.2 What is Essential?

The Information Technology (IT) world changes almost continuously as new technologies and techniques are adopted. Software testers (whether by title or in practice) must adapt quickly and be able to leverage their skills to meet new challenges. However, the essential skills and knowledge remain the same, serving as core understanding to which new information can be added. For the sake of readability, the term “software tester” will be used to refer to anyone who is testing software, regardless of their formal role.

This syllabus focuses on the essential areas of software testing that are required, regardless of the technology, lifecycle or tools in use. Some projects may use more or less of these skill areas, but all software testers need to understand and master this core skill set.

As the name indicates, this syllabus covers the “essentials”. This syllabus should be considered a springboard for additional certifications and knowledge areas. As a part of AT*SQA’s ISO compliant offerings, the certification must be kept current with additional learning completed within the defined timespan. For more details, see AT*SQA’s website. This helps software testers to continue to expand their knowledge and marketability and acknowledges the very real need for continuing education in the software testing industry.

0.3 Syllabus Structure

This syllabus has been constructed to be tool and methodology agnostic. In places where different approaches are needed based on different lifecycles, those areas are highlighted with appropriate recommendations for tailoring the approach.

The intended target audience for this syllabus is anyone conducting software testing, whether or not they have the title of software tester. This includes Scrum team members, developers, Business Analysts (BAs), software specialists and anyone interested in learning the important aspects of software testing.

This syllabus is intended to be read in full, but if the reader is interested only in a specific area, each area can be read independently. It is recommended that the Test Approach and Testing Techniques sections (Sections 2 and 3, respectively) are considered compulsory reading, as these are generally applicable to any of the specialist areas of testing and provide a good background to general testing practices.

0.4 Examinable Learning Objectives

Each chapter notes the time that should be invested in learning and practicing the concepts discussed in that chapter. This information should be used as a guideline when creating training materials or for an individual conducting self-study.

All identified key terms are examinable, either individually or by use within an exam question. Full definitions for the key terms can be found in the AT*SQA glossary (see www.atsqa.org).

The Learning Objectives for each chapter are shown at the beginning of the chapter and are used to create the examination for achieving the Testing Essentials Certification. Learning objectives are allocated to a Cognitive level of knowledge (K-Level). A K-level, or Cognitive level, is used to classify learning objectives according to the revised taxonomy from Bloom [Anderson00]. AT*SQA uses this taxonomy to design all examinations.

This syllabus considers four different K-levels (K1 to K4) as noted for each Learning Objective (LO):

K-Level	Keyword	Description
1	Remember	The candidate should remember or recognize a term or a concept.
2	Understand	The candidate should select an explanation for a statement related to the question topic.
3	Apply	The candidate should select the correct application of a concept or technique and apply it to a given context.

4	Analyze	The candidate can separate information related to a procedure or technique into its constituent parts for better understanding and can distinguish between facts and inferences.

In general, all parts of this syllabus are examinable at a K1 level. That is, the candidate will recognize, remember and recall a term or concept. Other specific learning objectives are shown at the beginning of the pertinent chapter.

1. Introduction to Software Testing – 60 mins.

Keywords

requirements, test case, test condition, test plan, test strategy

Learning Objectives for Introduction to Software Testing

1.1 What is Software Testing

LO-1.1.a (K2) Summarize the various forms of requirements
LO-1.1.b (K1) Recall the meaning of “fit for purpose”

1.2 A Brief History

LO-1.2.a (K1) Recall the difference between a test engineer and a test analyst

1.3 Structured Testing

LO-1.3.a (K2) Explain the purpose of the documents used in a structured testing environment

1.4 The Role of a Tester

LO-1.4.a (K1) Recall who can be a software tester

1.1. What is Software Testing

Software testing has variable meanings. The term has evolved as new software development lifecycle (SDLC) models have been introduced. Regardless of the changes to the exact definition, software testing is an activity, or set of activities, that are conducted to evaluate software to determine the following:

- Have the requirements been met?
- Is the software “fit for purpose”?
- Has the risk been reduced enough?
- Have important defects been identified and addressed?

Each of these questions tends to elicit more questions.

1.1.1. Requirements

Software requirements come in many forms including:

- Formal requirements documents prepared by Business Analysts (BAs)
- Technical requirements documents, such as functional specifications, design documents, and interface design documents

- Higher level documents, such as use cases which describe how an expected user would accomplish tasks or goals by using the software
- SDLC unique documents, such as user stories in the Agile lifecycle model
- Very informal diagrams on white boards and results from workshops
- Word-of-mouth and drawings in a highly collaborative environment (where the team is all working together, all the time)

The ability to verify that the software meets the requirements is dependent on the clarity of the requirements. If a requirement is clear and defines exactly what the software is supposed to do, the verification is straightforward. Where the requirements are vague or missing, the tester must be able to apply their own knowledge of the users and domain in order to determine if the requirements have been met.

1.1.2. Fit for Purpose

All software is designed to fulfill a purpose, but just accomplishing a task is not enough. In order for it to be "fit for purpose", the software must work for the people who will be using it, in the environment in which they will be using it. For example, a mobile application that allows people to deposit checks by taking a picture of the check may work great in the lab with specific lighting and backgrounds, but may fail when used in a user's home. In this case, the requirement may be met (it works functionally), but it is not "fit for purpose" because it is not usable in the target environment.

1.1.3. Risk

Because there is rarely enough time to perform all the testing possible, risk prioritization is used to limit the testing to what is needed to mitigate risk to an acceptable level. Determining what is acceptable may be a matter of opinion, which is why risk analysis requires cross-functional input to ensure each risk is being considered and rated accurately. With the above example of the check deposit, if the decision is that a low-lighting environment is highly unlikely, that would reduce the rating of that risk. On the other hand, if it is determined that this is highly likely to occur and that the user will be unable to deposit their check, the risk would be considered as very high and additional work would be required to adequately mitigate that risk. Risk is discussed further in Section 2.6.

1.1.4. Finding Defects

One of the purposes of testing is to find and fix defects before the software is released to the users. Defects, also called bugs, are flaws in the software that cause it to function incorrectly or cause the user to use it incorrectly. Clear requirements help in determining what is a defect and what is not. The less clear the requirements, the more discussion will be needed to determine if an anomaly is actually a defect or if it is just an undocumented feature of the software. Keeping the user's view in mind when testing the software helps the tester to better determine what a user would consider to be a defect. For example, an incorrect text prompt "enter suer name" is clearly a defect. What if the user name always has to be between 5-15 characters but the user is not told that? Is that a defect? Defect identification and proper recording is

an important task for a tester. Defects that are not recorded accurately are difficult, if not impossible, to fix.

1.2. A Brief History

Software testing has existed for as long as there has been software. The formality, emphasis, funding and respect for software testing has varied over the years, but it will always be needed. Good practices that were popular in the 1970's still have merit today, just as new practices developed since that time also have merit. It is important to remember that there is a wealth of knowledge in software testing. Environments, languages, devices and approaches may vary, but understanding the essentials of software testing will allow the tester to work in, and adapt to, any environment.

In software testing, there tends to be a differentiation between technical testers (i.e., test engineers) and non-technical testers (i.e., test analysts). Technical testers are expected to have the skills such as those needed to write test automation, conduct performance tests or participate in code/design reviews. Test analysts are generally expected to conduct the functional testing (i.e., does the software meet the requirements), as well as to consider usability (i.e., will the target user be able to use the software effectively, efficiently, and enjoy using it) and domain/environment attributes of the software. In some cases, test analysts are also expected to work with end-users for user acceptance testing (UAT) and to help validate that the software will work in the target environment for target users who are accomplishing the target tasks.

Like software development, software testing will continue to evolve. Mastering the essentials of software testing will help make a tester resilient and able to adapt to changes.

1.3. Structured Testing

Highly-structured testing, such as that required by some sequential lifecycle models (discussed in Section 2.3), generally has a higher level of documentation. Formal test strategies, well-defined test plans, explicit test cases, controlled test data and test environments, and a well-managed defect lifecycle are all artifacts of a highly-structured approach to testing.

While the documents may vary depending on the environment, the following are normally found in a structured testing environment:

- Test strategy – a test strategy is an organization-wide document that defines how testing will be conducted across all comparable types of projects in the organization.
- Test plan – a test plan is the implementation of the test strategy for a particular project and includes the approach to be used for testing, a definition of the scope of testing for the project, the testing schedule, the resource requirements, a description of tools and their usage, a definition of

environments and any other information required to describe the testing process, and stakeholder agreement for a project.

- Test conditions – a test condition is a capability or characteristic of the software that needs to be tested. This could be something functional, such as the ability to enter a user name; or something non-functional, such as the expected response time of the application under a defined load.
- Test case – a test case is the information required for a tester to test a test condition. This can include the pre-conditions of the system (e.g., user does not exist), the post-conditions after the test (e.g., the user has been created) and the inputs and actions required to accomplish the goal of the test.
- Defect reports – each defect should be captured in a report that is then processed through a workflow to record all the actions taken to resolve the issue. A defect report normally records information, such as the environment used, steps to reproduce, priority/severity, expected/actual results and other descriptive information.

More information about the documentation used in testing can be found in Section 2.5. Depending on the environment, more or less of these documents will be prepared and maintained as part of the testing process.

1.4. The Role of a Tester

The role of a "tester" can vary with different organizations and different lifecycle models. While software testing is a profession, others may periodically carry the title of a software tester. For example, in an Agile lifecycle model, everyone on the team has testing responsibilities and may be considered to be a tester. Business users may become testers during UAT. Software developers are testers when they are testing their own or another developer's code.

Regardless of the name of the role, testers are responsible for gathering information that can be used to assess the quality of the software. This information includes tests that have been run and have met their goals (passed), tests that have not met their goals (failed), defects found, risks mitigated, test coverage (in terms of tests executed vs. not executed, code covered vs. not covered, risks mitigated vs. not mitigated, or requirements tested vs. not tested) and other information needed by the stakeholders.

All testers need to be familiar with the essential areas of software testing. Specialization in these areas may require further study, but a general familiarity is necessary to understand what can and should be tested for any software product.

7. Usability Testing – 150 mins.

Keywords

accessibility, personas, usability, user experience, user interface, Web Content Accessibility Guidelines (WCAG)

Learning Objectives for Usability Testing

7.1 Introduction

LO-7.1.a (K2) Explain the difference between efficiency, effectiveness and satisfaction in usability testing

7.2 Focusing the Usability Testing

LO-7.2.a (K1) Recall sources for determining the needs and expectations of the users

LO-7.2.b (K2) Explain why depth or breadth testing would be used

7.3 Usability Test Participants

LO-7.3.a (K1) Recall the roles of professional testers and end-users in usability testing

7.4 Usability Test Planning and Design

LO-7.4.a (K1) Recall the use of personas in usability testing

LO-7.4.b (K1) Recall the purpose of user experience evaluation

7.5 Scheduling and Conducting the Tests

LO-7.5.a (K2) Explain when usability testing should occur in the SDLC

LO-7.5.b (K2) Describe how usability tests are conducted

LO-7.5.c (K2) Explain how results are gathered from usability tests

7.6 Standards

LO-7.6.a (K1) Recall standards that support usability testing

LO-7.6.b (K1) Recall the components of the usability software quality characteristic

7.7 Accessibility

LO-7.7.a (K1) Recall the meaning and purpose of accessibility testing

LO-7.7.b (K2) Summarize the common references and standards for accessibility testing

7.1. Introduction

Usability and accessibility of software are important quality characteristics and may determine whether or not the software is successful. Usability testing is conducted to evaluate how well a system can be used by the target users to accomplish a specified goal. Accessibility testing, which is considered a subset of usability testing, is conducted to ensure that users of all abilities can successfully use the product.

Efficiency (how much effort is required to accomplish a goal?), effectiveness (is the desired result achieved?) and user satisfaction (is the user “happy” with the software?) are factors that are often considered during usability testing. In addition, proper accessibility testing may be required as part of the overall testing project if accessible software is mandated by law. The results of both types of testing can then be used to improve the design to better meet the needs of the users.

The timing and scope of usability and accessibility testing can vary widely from project to project and, depending on the areas of a product targeted for evaluation, may be difficult to schedule and fit into the project lifecycle. Because of this, it is important to understand the goals of the tests before starting, consider when the testing should occur, and verify that there is adequate time in the schedule to plan the tests, review the results, and consider what changes should be made to the product.

7.2. Focusing the Usability Testing

In order to conduct effective usability testing, the tester must understand the needs and expectations of the users. This may include gaining an understanding of the product itself, as well as the day-to-day activities of the users. This information may be gathered from the following sources:

- Process maps
- Business cases
- Use cases
- Interviews
- Observation
- User guides and documentation
- Expert users or domain experts

Usability testing should concentrate on areas of the software that will be most frequently used and are most important to the user. If there is an existing product to use for comparison, complex user interfaces and existing features with a history of a high number of technical support issues should also be evaluated, to avoid perpetuating any problems.

Prioritization is important because the scope of usability testing is often limited by project schedule and budget. For new software, where usage is anticipated but not known, sampling may be done to allow testing across the overall User Interface (UI).

This type of testing will provide information regarding the expected user experience and can serve as a breadth-based test. For areas where extensive usage is expected, depth-based testing is appropriate. By approaching the testing with a calculated combination of breadth and depth testing, the best coverage can be obtained in the shortest period of time.

7.3. Usability Test Participants

In larger organizations, usability is a specialized discipline that leverages usability experts with training in psychology and design. Ideally, these experts also have some knowledge of software development and at least some exposure to testing. When professional testers conduct usability testing, they often work closely with usability experts to design tests and check for standard usability characteristics (e.g., navigation, number of mouse clicks to accomplish a task, or screen layouts). This type of controlled testing should be augmented with testing by real (or potential) users because users know what they expect the software to do, what they need it to do and how they expect it to work.

For a given project, the usability testers should represent the skill levels and knowledge of the actual users. If there will be Subject Matter Experts (SMEs) using the software, they will have a different approach than a novice user. Both of these user types should be considered during usability testing.

While both types of testing are important, the feedback from real users is sometimes given more credence than feedback from professional testers, since the users often have actual experience with the product. This is particularly true when an observation is subjective (e.g., “the layout is confusing”).

7.4. Usability Test Planning and Design

Test planning is recommended for usability testing, in order to define the test approach, identify the goals, and to gain agreement among different stakeholders regarding the scope and expectations. With formal usability testing, guidelines for observer behavior are usually described in the test plan. Observers may be expected to not interfere, to provide help only when requested or to step in when the user appears to be getting frustrated.

As part of usability testing, users are often categorized into personas. A persona is a representation of a type of user and is often given specific characteristics (such as age, gender, profession, etc.). These personas are used during usability design to help ensure that the needs and desires of all targeted user sets are met. During testing, the activities and reactions of real people can be evaluated against the personas that were used during design.

Another aspect of usability testing is user experience (UX) evaluation, which should also occur as early as possible. User experience refers to a person’s perceptions of

and responses to the software before, during and after usage. For example, happy anticipation of the use of the software is a UX characteristic. Brand image and presentation of the software contribute to user satisfaction. Like usability, the user experience must be designed into the software in order to be achieved in a cost-effective manner.

7.5. Scheduling and Conducting the Tests

7.5.1. Early and Continuous Testing

As with all forms of testing, it is less expensive to correct usability issues when they are uncovered early in the SDLC. Early testing is often conducted as the software is being designed (sometimes called formative testing as the software is being "formed"). This allows users and specialists to review the design before it is coded. It may be done with written descriptions, wireframes (low fidelity) or prototypes (high fidelity).

Ideally, usability testing is a continuous activity, performed frequently during software design and development. This will provide timely feedback and will help influence the design as the project progresses. In an Agile methodology, the product owner normally assists with usability testing during iteration testing to ensure that the product meets expectations.

7.5.2. Conducting the Usability Test

Formal usability tests are typically conducted in a usability lab, which may be equipped with video and audio recording devices, two-way mirrors and a separate seating area for observers. A formal lab may have software installed which tracks eye movements of the subjects and records the interactions of the subject with the product. Less formal testing may be conducted in an office setting or even a test lab environment. Ideally, the tests are conducted in an environment that closely mimics the user's work environment. This will provide a more realistic experience (e.g., lighting, noise, and distractions) and will help the user to better evaluate the software.

Usability tests are sometimes conducted by having the subjects perform tasks that have been designed in advance by a usability tester. Testing may also be conducted by assigning general tasks to the users and having them figure out how to accomplish those tasks on their own. User manuals and process documents may be used as guidelines for testing and the testing is sometimes used to review the correctness of the documents.

During usability tests, subjects generally are asked to vocalize what they are thinking while they are working through their tasks (i.e., think out loud), express any confusion that they may have and talk about what they are doing. If observers are present in the room during the usability tests and are required to be silent and passive, they can give no help or feedback to the participants. This is an important consideration as it is sometimes difficult for the observers to avoid influencing the tests.

7.5.3. Gathering Results

Feedback from the tests are usually gathered in two ways: defect reports and questionnaires/surveys.

Where usability or user interface defects are identified, the normal defect lifecycle should be followed, with particular attention paid to maintaining consistency. For example, if users object to the way a button is displayed, all buttons should be reviewed to determine if broader changes are needed.

Questionnaires and surveys are used to gather feedback regarding the effectiveness and efficiency of the software and the user's satisfaction with their experience. These surveys may also extend into the UX areas (e.g., emotions, perceptions, preferences, image).

7.6. Standards

There are several international standards that deal with usability. The following are commonly used:

ISO 9241-210 discusses human-centered design. This is based on understanding the expected use, specifying requirements, producing solutions, evaluating the solutions and eventually designing the best solution to meet the usability requirements.

ISO 25010 describes the software quality characteristic called usability. This breaks usability into a set of characteristics as follows:

- Appropriateness recognition – Can the user determine if the software is appropriate for their needs?
- Learnability – Can the user figure out how to accomplish a task and are they able to apply that knowledge the next time they want to accomplish the same or a similar task?
- Operability – Is the software easy for the user to operate and control?
- User error protection – Does the software help prevent the user from making errors?
- User interface aesthetics – How pleasing or attractive is the software to the user?
- Accessibility – Can the software be used by people with a wide range of capabilities?

These two standards help to guide usability design, as well as usability testing.

7.7. Accessibility

Accessibility testing is considered a subset of usability testing. Accessibility is the degree to which a component or system can be used by people with the widest range of characteristics and capabilities to achieve a specific goal in a specified context of

use. While sometimes targeted at specific disabilities, such as color blindness or hearing impairment, accessibility has generally become a broadened concept to ensure that software works for everyone, with or without disabilities.

Accessibility compliance requirements may drive accessibility testing goals and methods. It is important to clearly identify any pertinent legislation or regulations that apply, such as the ADA (Americans with Disabilities Act) and Section 508, before planning the testing, as specific goals may be defined in the regulations. Section 508 Compliance Testing, an amendment to the United States Workforce Rehabilitation Act of 1973, is a federal law mandating that all electronic and information technology developed, procured, maintained, or used by the federal government be accessible to people with disabilities.

An internationally used reference for accessibility testing is the Web Content Accessibility Guidelines (WCAG). These are widely used guidelines that were published by the Web Accessibility Initiative (WAI) or the World Wide Web Consortium (W3C). There are three conformance levels defined in WCAG: A, AA, and AAA, with AAA being the most difficult to achieve [WCAG].

Accessibility testing tools are available and can be used to quickly scan code to identify compliance issues. These tools will look for such items as text descriptions for all graphic items, usage of colors and font sizing. The tools are frequently updated and provide coverage for different accessibility areas. Research is needed to find the best tool for a particular situation. Accessibility tends to be a specific area of testing expertise because it requires a deep understanding of the regulatory requirements and the tools that will determine conformance to the standards.

10. References

10.1. ISO/IEC/IEEE Standards

- ISO/IEC/IEEE 12207:2017
- ISO/IEC/IEEE 15288

10.2. Trademarks

The following registered trademarks and service marks are used in this document:

- AT*SQA® is a registered trademark of the Association for Testing and Software Quality Assurance

10.3. Books

[Anderson00]: Anderson, L.W. and Krathwohl, D.R. (2000) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon: Boston MA, ISBN-10: 080131903X

[Firtman]: Maximiliano Firtman, "Programming the Mobile Web", O'Reilly Media; Second Edition (April 8, 2013), ISBN-10: 1449334970

[PMBOK] Project Management Institute, "A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Sixth Edition, 2017, ISBN-10: 9781628251845

10.4. Other References

The following references point to information available on the Internet. Even though these references were checked at the time of publication of this syllabus, AT*SQA cannot be held responsible if the references are not available anymore. AT*SQA is not endorsing any of these sites or their products. The references are provided as a source of information only.

https://techcrunch.com/2013/03/25/ip-oh-my-gosh-all-that-money-just-disappeared/

https://www.reuters.com/article/us-facebook-settlement/facebook-settles-lawsuit-over-2012-ipo-for-35-million-idUSKCN1GA2JR

[NASDAQ] https://www.sec.gov/news/press-release/2013-2013-95htm

National Institute of Standards and Technology. Framework for Improving Critical Infrastructure Cybersecurity. Version 1.1. 2018. https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf

National Institute of Standards and Technology. Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy. Revision 2. 2018. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-37r2.pdf

[WCAG] https://www.w3.org/WAI/policies/