AT*SQA

MICRO-CREDENTIAL

Testing for IOT and Mobile

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.

8. IoT (Testing Connected Devices) – 180 mins.

Keywords

connected devices, emulators, lightweight testing, simulators

Learning Objectives for IoT (Testing Connected Devices)

8.1 Introduction

None

8.2 Connected Devices

LO-8.2.a (K2) Understand the challenges with connected device testing

8.3 Environments and Tools

LO-8.3.a (K2) Understand the differences between simulators and emulators, and which is appropriate for a given situation

LO-8.3.b (K1) Recall how test automation tools can assist in connected device testing

8.4 Quality Characteristics

LO-8.4.a (K1) Recall why understanding user expectations is particularly challenging for connected devices

LO-8.4.b (K1) Recall why requirements for quality characteristics must be identified early in the lifecycle

LO-8.4.c (K2) Explain why usability is an important quality characteristic to consider in connected device testing

LO-8.4.d (K2) Explain why performance is an important quality characteristic to consider in connected device testing

LO-8.4.e (K2) Explain why security is an important quality characteristic to consider in connected device testing

LO-8.4.f (K2) Explain why interoperability is an important quality characteristic to consider in connected device testing

LO-8.4.g (K2) Explain why accuracy is an important quality characteristic to consider in connected device testing

LO-8.4.h (K2) Explain why reliability is an important quality characteristic to consider in connected device testing

8.5 Lightweight Testing

LO-8.5.a (K2) Describe how lightweight testing can be advantageous in connected device testing

8.1. Introduction

The world of connected devices is expanding. What started with mobile phones soon became smart phones and tablets, then blossomed into the Internet of Things (IoT). As software has adapted to be quicker and smaller, testing must also adapt to be quick and lightweight. That said, good testing practices still apply and quality characteristics are still important to the users. However, it is important to remember that users have expectations that their mobile applications and IoT devices will "just work".

8.2. Connected Devices

Devices in the connected world vary from smart phones to refrigerators, from tiny humidity detectors to cars. Anything that is capable of supporting an Internet enabled component is capable of joining the IoT. The same software may be supported on a variety of devices and operating systems (OSs), making compatibility testing more challenging and future-proofing difficult. As the industry leaps forward, backward compatibility is often receiving less emphasis when actually more is needed. There is an expectation from users not to be forced to upgrade in order to take advantage of new features and applications.

Testers can no longer expect to have access to all the devices that will use the software under test. Selecting a representative sample set is a critical part of defining test coverage and risk mitigation. Just because the software runs on a refrigerator allowing the user to increase or decrease the temperature remotely does not mean that all models of refrigerators need to be tested. It is important for testers to understand what exactly is to be tested. Is it the device that provides the connectivity? Is it the response of the target device? Is it the communication between the two? Realistically, it is all of these, but if all refrigerators use the same communication interface with the Internet device, then testing one may be sufficient (i.e., applying equivalence partitioning).

8.3. Environments and Tools

The proliferation of devices supported by Internet appliances has resulted in a plethora of tools and simulators/emulators that can be used for testing. This reduces the need for having a large set of devices available for testing and can greatly speed up manual testing and the development of test automation. Device labs are available for popular devices, such as smart phones, and new simulators/emulators are constantly being created. It is always good practice to verify the results from a simulator/emulator against a real device, but the majority of the testing does not require the more expensive real devices.

Simulators generally provide a standard response to various inputs and “simulate” the interaction with a real device at the software level. Emulators go a step further and “emulate” the responses of the hardware device as well as the software running on that device. For example, testing an application’s interactions with a smart phone’s gyroscope requires an emulator, whereas testing interaction with the device’s email application can be done with a simulator. Similarly, simulators in the form of service virtualization can be used to simulate a web service’s interaction with an application.

Test automation tools are quickly adapting to the IoT and mobile device market. Many of these tools will interact with simulators and, sometimes, even include their own simulators. Simulators for compatibility testing, such as cross-device or even cross-browser, are commonly supported by test automation tools. The tendency is for these automation tools to be more lightweight in features than the traditional client/server or web services tools. Cost is always a consideration and the open source market quickly adapts to the needs for new and purpose-built tools.

Even the best simulators/emulators cannot simulate/emulate everything. There are user actions, such as a complex set of gestures, that must be tested on a real device. It is important for the tester to determine which tests are best conducted with which environment. Generally a mix of simulators/emulators and real devices will yield the most accurate result for the least cost.

8.4. Quality Characteristics

While quality characteristic testing is important for all types of software, there are some unique needs for quality characteristic assessment when dealing with connected devices. One of the most difficult aspects of connected device testing is defining the users’ expectations. The user group for an application can be quite large and the expectations may vary dramatically, even within the target users. It is particularly important that the requirements for the quality characteristics are clearly defined in a measurable way in the requirements or acceptance criteria for the software. With all quality criteria, there is a range of “acceptable”. Defining and documenting this range early in a product’s lifecycle is particularly important for connected devices. By defining the criteria for the quality characteristics at the beginning of the project, all architecture, design and implementation decisions can be made to align with and fulfill those objectives.

While all quality characteristics are important, the following characteristics are particularly important with connected devices:

- Usability
- Performance
- Security
- Interoperability
- Accuracy
- Reliability

8.4.1. Usability

As connected devices become more integral to modern life, users expect "good" usability and learnability. Particularly for downloaded applications for smartphones, users expect software to be attractive, inviting, easy to use, easy to understand, and enjoyable. Users are becoming less patient with software that does not provide a good user experience. This can result in a product that is functionally sound being rejected by the users because it does not provide the expected feedback (e.g., prompts and sounds). Of course, being functionally sound is still the basic building block of software quality, but the ease with which a new user can and wants to work with an application often determines the market share. See Section 7 for more about usability.

8.4.2. Performance

Performance criteria tend to be loosely defined and are often identified when a product does not meet expectations. While this is true for any software product, it is apparent for applications for connected devices which may have limited capability (memory, bandwidth). Ideally, the performance criteria should be defined and captured early in the requirements phases of a project. Defining these criteria, setting up the proper personas and benchmarks, and testing to ensure the criteria are met on the target device, are critical for the success of a connected product. It might be acceptable for a refrigerator to be slow to respond to a request to decrease temperature, but it is unacceptable for a GPS-based guidance application on a smart phone to not respond in time for a driver to make a turn.

Another variable with connected devices is the possibility of a lack of connectivity. This can significantly impact both functionality and performance. A poor connection can result in poor performance. A product may not be able to control the quality of the connection, but it can control its response to poor, intermittent or non-existent connections. See Section 5 for more about performance.

8.4.3. Security

Connected devices are sometimes used for safety-critical applications as well as for financial transactions. Cybersecurity testing is difficult with the shortened development cycles of connected devices, but it is a critical component of a quality product. Architecting, developing, and testing for security must occur throughout the development lifecycle, as there will rarely be enough time at the end of development for thorough cybersecurity testing (and any necessary corrections/changes).

It is important to understand the expected and potential usage of an application running on a connected device. Is the GPS-based guidance application used to get to the mall or to direct ambulances to emergency scenes? Is a web-monitored intruder alarm used to guard the refrigerator from marauding teenagers or to protect a pharmacy? Seemingly simple devices can easily be used in safety-critical situations, necessitating the highest levels of security testing. Like quality, security must be built into the product right from the start.

Connected devices have unique security risks. For example, an attacker can gain access through man-in-the-middle attacks by exploiting Bluetooth vulnerabilities of the device. Using connected devices with public Wi-Fi hotspots can lead to security vulnerabilities that would not be experienced on a controlled and protected network. Because mobile devices are easily carried with a person, they are also easily lost or stolen, which can allow personal information to fall into the hands of a criminal. Connected medical devices present a unique security risk as they have been compromised in past Wi-Fi cyberattacks to gain access to the larger network in a hospital. The list goes on and on. In general, security requirements for connected devices are always expanding, particularly as new vulnerabilities are discovered.

See Section 6 for more about cybersecurity.

8.4.4. Interoperability

Connected devices offer some unique challenges for interoperability. Software is often developed to work on multiple operating systems and a range of devices. For example, a banking application may be intended for use on a wide range of smart phones and tablets. The portability of an application to different devices, the ability of an application to interoperate with other software and the compatibility of the software across different browsers and operating systems are often lumped into the general category of interoperability.

From a testing perspective, this gives a nearly infinite number of combinations to test and those combinations continue to evolve rapidly. Using combinatorial testing techniques can bring this potential set of test targets down to a manageable number. This type of testing tends to be pushed to the end of the testing cycles, when there is enough functionality to support the testing.

The architecture of the product often determines its interoperability capabilities. If the architecture is limiting, the capability of the end product will also be limited.

8.4.5. Accuracy

Accuracy testing targets the ability of the software to provide accurate results for a given set of inputs. Users expect software to be accurate, but there are specific considerations for connected device software primarily because the usage can be so varied. A simple humidity detector developed on a Raspberry Pi may be intended for use by plant enthusiasts to ensure proper watering levels. However, this same detector could be used to ensure an asthmatic child has the proper humidity in their room air. Because the use of a product may not be controllable, accuracy must be ensured for any possible safety-critical usage. This tends to push most of the connected device testing into the safety-critical arena, unless a product can be specifically excluded from a safety-critical use.

Testing in a safety-critical environment requires higher levels of documentation and due diligence to protect the developing organization from potential legal action if something should go awry. This is an area that is open to debate, but from a testing

standpoint, more thorough and better documented testing will help mitigate deployment risk and may help meet regulatory and industry standards. This is particularly challenging in the short cycles of connected device products and is why lightweight testing approaches are critical.

8.4.6. Reliability

As with the higher expectations for usability and performance, users expect complete reliability from their connected devices and software. No one expects to reboot their smart phone and much less their refrigerator. Realistic or not, this is the expectation that must be met in order to capture and retain users. Testing for reliability is difficult and requires test environments that are representative and stable. It also requires time, as reliability tests usually require applications to be observed under continual use for a given length of time to determine the mean time between failure (MTBF). The time required for this testing potentially conflicts with the goal of being quick to market.

Reliability cannot be tested into an application – it must be built in. This means that reliability targets must be set early and reviewed frequently. Reliability testing in the connected device area tends to be limited to discovering whether there are significant problems. More subtle problems or problems that only appear over long usage tend to be discovered in production. Perhaps more than any other area, reliability assurance is a development activity more than a testing activity.

8.5. Lightweight Testing

A common theme has emerged in this section – software has to be built right because there is no time to fix it later. This means the requirements for quality characteristics must be defined and understood by the development and testing teams. In lightweight testing models, such as Agile, the whole team approach helps everyone to review and understand these requirements from the beginning and to revisit and validate fulfillment of the requirements throughout development. Any lightweight methodology is dependent on the engagement of the BAs, product owners, developers and testers throughout the lifecycle to ensure that the product is reviewed and tested as it is being built. Testing cannot be pushed to the end of the process if the schedule time is to be reduced and quality criteria met.

Testing documentation must be as lightweight as possible in this environment. This means that detailed test cases are probably not needed, but repeatable tests are. Testing from decision tables and checklists rather than from step-by-step test cases is a good way to test the implementation of business processes and workflows. The testing techniques can be well-applied to reduce testing documentation while increasing coverage and repeatability. Exploratory testing can help fill the gaps between the testing techniques but should not be the only form of testing, as repeatability tends to be compromised or lost. To adapt to this changing environment, testers need to document the minimum needed for repeatability and to let risk and coverage goals guide the testing.

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/