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.

5. Performance Testing – 200 mins.

Keywords

concurrency testing, load test, objectives-based reporting, operational profiles, performance testing, risk-based reporting, scalability, stress test

Learning Objectives for Performance Testing

5.1 Introduction

None

5.2 The Purpose of Performance Testing

LO-5.2.a (K1) Recall the goals of performance testing
LO-5.2.b (K2) Explain why defining and getting agreement on performance requirements is important
LO-5.2.c (K2) Explain how performance testing aligns with the SDLC

5.3 Performance Testing Risks, Benefits, and Challenges

LO-5.3.a (K1) Recall the unique challenges of performance testing
LO-5.3.b (K1) Recall the risks of performance testing
LO-5.3.c (K1) Recall the benefits of performance testing

5.4 Performance Testing Approach

LO-5.4.a (K2) Describe the components that should be included in a performance test plan
LO-5.4.b (K2) Explain the factors to be considered when defining the test approach

5.5 Conducting Performance Testing

LO-5.5.a (K1) Recall the steps for conducting performance testing
LO-5.5.b (K1) Recall the components that should be checked during performance test preparation
LO-5.5.c (K1) Recall how performance test results differ from functional test results
LO-5.5.d (K1) Recall the types of performance test reporting

5.6 Performance Test and Analysis Tools

LO-5.6.a (K1) Recall why tools are necessary for performance testing
LO-5.6.b (K2) Describe the challenges that can be encountered with performance testing tools

5.1. Introduction

Performance is a major quality factor in most systems and applications, regardless of the computing environment. A system or application may be functionally correct, but if it fails to deliver the needed performance it is likely to be considered a failure. Performance testing is one way to measure system performance in advance of deploying the system.

Performance testing should start at the component (unit) test level and continue until system deployment. Waiting until the end of a project to conduct performance testing carries the high risk that performance problems will not be solvable due to time, money and technological constraints.

One of the most costly and publicized performance failures occurred during the Facebook Initial Public Offering (IPO), when the NASDAQ stock exchange could not handle the volume of trading on the day of the IPO. The cause of the failure was determined to be three lines of code that got into an endless loop. To date, NASDAQ has paid over $80 million in fines and restitution. In addition, the United States Securities and Exchange Commission imposed requirements on NASDAQ to help prevent similar failures in the future [NASDAQ].

5.2. The Purpose of Performance Testing

Performance testing can have multiple goals and purposes, including:

- Measuring system performance under given conditions
- Determining the maximum concurrent user load or transaction volume a system can handle before it fails
- Providing information to assist in capacity planning for a system

Significant time and money can be invested in performance testing. In order to obtain the best return on investment, the testing must be targeted correctly and the goals clearly defined. This can be difficult since different stakeholders may have varying, even conflicting, views of “acceptable” performance. In order to be successful, there must be agreement in the project team regarding what the performance testing will measure and what will constitute a successful result.

5.2.1. Defining Performance Requirements and Getting Stakeholder Agreement

Documenting performance requirements can be challenging. Eliciting the requirements can be even more difficult. When documented, performance requirements are normally recorded in over-arching requirements, such as “all system responses to users that require longer than 3 seconds must display a wait notification”. In an Agile environment, these requirements may be documented in specific non-functional Epics.

Without stakeholder performance requirements, it is hard to know if the observed levels of performance are adequate, making it difficult for testers to make a reasoned assessment of test results. A complicating factor in determining system performance requirements is that the assessment of “acceptable” performance can be subjective.

Getting agreement on performance requirements can be quite challenging because of the subjectivity of opinions and the cost required to achieve higher levels of performance. For example, one stakeholder group might feel that a response time of two seconds is acceptable, while another group might feel that a response time of one second or less is required. The cost to achieve the one second response time might be quite high (e.g., 5x improvements in hardware, more robust networks, major modifications to code or databases), whereas two second response time may be easily achieved.

5.2.2. Aligning Performance Testing in the SDLC

As with all forms of testing, performance testing needs to be integrated with the SDLC. Regardless of the SDLC, full lifecycle performance testing is needed to detect performance problems early, when they are easiest to fix.

The following activities in an SDLC have key tasks for performance testing:

- During requirements definition, the specific requirements for performance (and definition of the expected load) should be clearly defined and agreed to by all stakeholders. This will eliminate debate when the results are reviewed later.
- During design and coding, performance factors must be considered and built into the design and the subsequent code. Inefficient code or an inefficient design will be difficult and costly to fix later.
- During integration and system testing, new performance testing opportunities can appear. Any new integration may introduce inefficiency and potential performance bottlenecks.
- During acceptance testing, a major objective is to assess performance within the business or operational context. In some cases, performance testing is part of contract acceptance testing and operational acceptance testing.

5.3. Performance Testing Risks, Benefits, and Challenges

Performance testing has unique challenges, primarily due to lack of understanding of the complexity and cost of good performance testing. These include the following:

- Getting definition and agreement from stakeholders with regards to acceptable system performance
- Getting adequate funding for performance testing tools
- Acquiring the best fit solution for a performance testing tool, within the schedule and budget of the project
- Accurately profiling load levels at given periods of system usage

- Acquiring skilled test engineers to plan, design and conduct a realistic performance test
- Building a representative performance test environment

Unless these challenges are met and understood, the performance test effort may never start or will not be effective. In addition to the above challenges, performance test efforts have risks that must be considered and mitigated in order to achieve the expected benefits.

5.3.1. Risks

Performance testing has the following risks:

- Inadequate performance test design, resulting in inaccurate or incomplete test results
- Incomplete coverage of protocols and connectivity, resulting in important aspects of performance testing for a particular system or application being missed
- Inadequate test environments, resulting in inaccurate test results and erroneous conclusions
- Inability to apply the performance tool correctly, resulting in inefficient, inaccurate, and inconclusive tests
- Inadequate coverage of functions, resulting in incomplete test results
- Inadequate amounts of user and data load, resulting in inaccurate and incomplete test results
- Lack of defined stakeholder requirements for performance, resulting in the inability to identify performance targets
- Lack of agreement on defined stakeholder performance requirements, leading to conflicts concerning acceptable levels of system performance
- Lack of appropriate performance metrics, resulting in test reports that are incomplete or lack depth of meaning.

5.3.2. Benefits

If the challenges are met and risks mitigated, significant benefits can be expected from performance testing. These include the following:

- Opportunities to “right-size” the system to handle expected load
- Opportunities to plan mitigation steps if loads exceed expected levels
- Early test results can help define the acceptable levels of performance
- Expectations for future growth can be tested to know if the system will be able to support it
- System performance weaknesses can be identified and fixed before being discovered in production
- Long lead time items, such as adding more hardware or improving network resiliency, can be addressed in adequate time before a production launch

Because performance issues may be expensive or time-consuming to rectify, identifying them early on allows the greatest chance for mitigation before an exposure

in the production environment. Performance testing is often intended to merely confirm expectations that the overall system is fit for purpose, but frequently results in identifying significant problems that must be corrected.

5.4. Performance Testing Approach

Performance testing often requires a separate planning effort. This effort is focused on the particulars of the performance testing, such as sizing the test systems, procuring resources and planning uninterrupted testing time.

5.4.1. Defining the Test Plan

Performance test plans generally include the following information:

Scope

The correct setting of scope in performance testing is a critical activity to ensure that the targeted objectives are met. The scope is often framed by the following:

- Functionality – What features will be included or excluded from the performance testing?
- Architecture – Which aspects of the system architecture (e.g., networks, APIs, devices, databases) will be included?
- Transactions – What typical user transactions will be included in the testing?
- Users – Which classes of users should be included based on selection factors (e.g., location, type of transactions, demographics, concurrent usage)?
- Data – How much data should be used and how should it be accessed?

Strategy and Approach

The performance test strategy and approach can define the way performance testing is to be conducted. For example, an organization might decide to outsource performance testing if it lacks the tools and skilled people to conduct the testing on its own.

Risk Assessment

As in much of testing, performance testing can also be risk-driven to focus testing on areas of the system, or on functions and transactions, that carry the highest risk, such as those with the highest use.

Definition of Test Objectives

Performance test objectives describe, at a high level, what the performance test is intended to achieve. Oftentimes, these are worded as performance objectives to verify and/or validate. For example, “Verify the system response time is 1 second or less at peak load times.”

Responsibilities

Team roles must be defined, along with roles and responsibilities for those external to the team and the organization. Performance testing often requires the support of

system architects, database analysts, network engineers and other specialists who can participate in troubleshooting and system tuning.

Reporting Metrics
Metrics have a very important role in evaluating performance test results. It is important to define which metrics are most meaningful to stakeholders – both business and technical – and ensure those metrics are tracked by the selected monitoring tools. Some metrics, such as average response time, may not be meaningful or as useful as the response time for 90% of the users. This will help eliminate any outliers that can skew the average.

5.4.2. Defining the Test

Depending on the formality of the environment, the test approach may be defined as part of the performance test plan or defined separately. The test approach must consider the following:

Environments
Test environments are a key concern in performance testing because of the need to obtain representative results. The challenge is that full-scale dedicated test environments are often difficult to build due to cost and complexity. One possible alternative is to use a cloud-based test environment that can be rapidly scaled to simulate production configurations.

Unfortunately, it is not wise to simply predict performance at higher levels of scale based on lower system capacities. It is normally necessary to apply realistic load with realistic system conditions.
In some cases, such as in new system development, the system under development can be tested in the same system configuration that will eventually become the actual production system.

Load and Throughput Profiles
To accurately design performance tests, current and predicted profiles must be understood for both load levels and data throughput, such as peak load times and load levels. A common way to know performance profiles is to measure and study user behavior and levels as captured in system analytics.

Operational Profiles
Operational profiles describe the functions users are expected to perform on the system and the frequency of these activities. These can be seen as simple functions or wider-ranging workflows for end-to-end transactional testing of the system, sometimes called “transaction threads”. Operational profiles are implemented as scripts to be executed by virtual users, creating an accurate environment for measuring performance.

Test Data

Representative amounts and types of test data are often at the heart of performance testing. This data may be generated by tools or copied from a production environment if no private data is involved.

5.5. Conducting Performance Testing

In order to run the performance test, several other steps are needed. The tests must be prepared (including the environment and data), they must be executed, the results must be evaluated and reported, and there may be a need for on-going monitoring.

5.5.1 Test Preparation

Prior to test execution, the following components should be verified as ready for the testing:

Test Environment
It is often helpful to run preliminary tests to make sure basic test environment elements are in place and working correctly and that proper access is available. It is also important to validate that all co-existing applications and systems that could cause a performance load are in place and operational, as these systems can also place performance load on the test environment.

Test environment capacities need to be verified as correct. For example, CPU levels, database size and network configuration all have an impact on system performance.

Data
All databases and files should be populated with the designed volumes of test data to ensure the tests are not blocked due to insufficient or incorrect data. Generating this data and verifying its correctness must be conducted prior to performance test execution.
Testware (Test Scripts, Procedures, etc.)
As part of test preparation, it is vital that the designed and implemented testware is in place and executing correctly as designed. An effective activity is to conduct a preliminary test using a representative sampling of test procedures to verify functional correctness and data accessibility and accuracy.

5.5.2. Test Execution

If the preparatory steps have been completed, then test execution becomes a matter of running the tests in the tool and monitoring the execution. Monitoring features are a part of many performance test tools. However, the interpretation of test results requires human intelligence.

Sometimes, it is only by observing test results in real-time that some performance anomalies may be identified. For example, it is common in a performance test to ramp up load levels. In some test tools, the impact of increasing system load can be seen by analyzing graphical representations after the test is complete. However, other tools

may primarily display test results in a text-based format. In those cases, it is helpful to observe system behavior and metrics as the test is in progress.

5.5.3. Test Evaluation

In contrast to functional testing, where test outcomes can be evaluated as “pass” or “fail”, performance test results tend to be more comprehensive and informative. The main question that performance testing seeks to answer is “Does system or application performance meet stated performance goals or requirements?”

It is important to understand that performance testing is highly dynamic and dependent on many factors, such as user load, workflows performed, system capacity (CPU speed, database efficiency, code efficiency, and networking configuration), and system configuration. Changing any of the variables can significantly affect the results.

Performance testing is a snapshot view. Even a small change to the system can cause improvements or degradation of performance. For this reason, performance testing should be performed throughout system development and after release to production.

Performance testing is sometimes used to assess the system’s capabilities. While load testing can determine performance levels at given load levels, stress testing can reveal the maximum capacity the system can handle based on a given configuration. This information can help capacity planners to know if system upgrades will be needed, and if so, when they may be needed.

5.5.4. Test Reporting

The main deliverable of performance testing is the reporting of the test results. Performance test reporting will vary depending on the audience and the purpose of the test. Since each stakeholder group has differing levels of technical understanding, test results need to be presented in ways that each group can understand and find meaningful.

Risk-based Reporting
As in other forms of testing, risk-based reporting can help identify which aspects of system performance may carry the most relative risk. This risk-based view of performance test reporting can help stakeholders understand where to focus efforts for the most effective system implementation decisions.

Objectives-based Reporting
In objectives-based reporting, performance test results are reported with traceability to performance test objectives, such as the expected response time to a specific user action. This allows testers and stakeholders to know whether or not performance objectives have been met as demonstrated by the success or failure of performance tests.

5.6. Performance Test and Analysis Tools

5.6.1. Why Tools are Essential for Performance Testing

Performance testing is one type of testing that is not feasible without tools. Tools are used for the following:

- Creating simulated concurrent user load – This simulated load is used for concurrency testing. It is more precise and does not require as many resources as trying to test with large numbers of people. Concurrency testing is used to determine how the system will perform with a consistent load of virtual users who are “concurrently” using the system.
- Sustaining high levels of load – Even if it was possible to generate enough load with actual people, it would be difficult to sustain the load long enough to get a good measurement or to allow repetition of the test.
- Accurately measuring load levels and system response time, CPU utilization, memory allocation, etc. – It takes more than a stopwatch to measure response times. Most performance test tools have features to measure aspects of the SUT that directly impact performance. For tools that lack robust monitoring functionality, it is possible to obtain tools that independently monitor test performance.
- Repeating performance testing whenever needed – Performance tests must be repeated as changes are made to the application and system. Often, during performance testing, system tuning takes place to improve the performance. This requires re-execution of the tests to ensure that the expected improvement has been realized.

5.6.2. Tool Challenges

Although tools are needed for performance testing, there are some challenges to overcome:

- Performance test tools do not know what to test – The performance tester must determine which functions to test, how many concurrent users to simulate, which data to use, and other conditions to achieve the test objectives.
- The environment has to be representative – For many people, building a performance test environment may be the greatest challenge of all. Adequate performance test environments may require an investment in hardware, software, people, and tools – all scalable to the level of production use. Cloud-based virtual test environments have become an attractive alternative to physical environments in some cases.
- High volumes of test data will be needed – This may require the use of test data generation tools or strategies to modify a copy of existing production data.
- Expertise is needed – Performance testing requires skills and experience not commonly found in many organizations. It is common to obtain outside consultation when first starting to plan and conduct performance testing.
- Tools can be expensive – The most robust and popular performance test tools can be very expensive. While new tools in the marketplace help to keep pricing competitive, organizations that have large investments in performance test

tools may be less inclined to switch to more affordable tools. Even open source tools have a cost in learning and maintenance.

Selecting the proper tools requires evaluating a number of factors including long-term cost, training requirements, vendor reliability, etc. Because performance tools may be significantly expensive, it is important to consider the life expectancy of the tool, the ROI and the likely future requirements for tooling.

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/