BCS Professional Certificate – Software Tester, Supporting Content to Learning Objectives

Version 1.1
June 2020

Contents

Change History

Any changes made to the syllabus shall be clearly documented with a change history log. This shall include the latest version number, date of the amendment and changes made. The purpose is to identify quickly what changes have been made.

Version Number	Changes Made
V1.1 June 2020	Minor updates to some wording for clarity
V1.0 April 2020	Final
V0.2 Jan 2020	First draft of final Syllabus
V0.1 Nov 2019	Initial draft

Introduction

This document provides comprehensive supporting materials for the Learning Objectives in the BCS Professional Certificate - Software Tester syllabus and should be read in conjunction with that document.

1. Testing Activities in Development Lifecycles (20%)

Introduction

Building on foundation knowledge of testing activities in different lifecycles, a Professional Software Tester should be capable of analysing a typical project scenario and choosing appropriate activities for the situation. Acknowledging the trends towards multi-skilled staff, collaboration between those with different roles, together with the demand for technical skills applied in many testing activities, the scope of this topic area in the syllabus covers:

- Certain principles of software architecture that may impact on different levels and types of software testing
- Shared knowledge, skills and behaviours for software testing activities regardless of role and development lifecycle

Whilst the syllabus does not cover in any detail the different roles or skills within the development lifecycle, there is a level of technical awareness covered that is required for a modern Professional Software Tester to be able to do their job and meet the objective outlined for this topic, including test automation and certain quality characteristics.

Software architecture and development platform context

There are number of elements that contribute to the underlying architecture of software applications:

- Code and libraries
- Data
- Application components
- Application interfaces

Applications are characterised by the separation of software 'logical' layers responsible for performing certain types of tasks. These are outlined in the OSI (Open Systems Interconnection) model, but for the purposes of this syllabus we shall consider the software layers in the following typical summarised model:

- Presentation (e.g. A web page allowing a user to access a system)
- Business logic (e.g. The execution of functions to perform a specific business feature)
- Data (e.g. The storage and retrieval of information in data sources)

In this context, the presentation layer is known generally as 'front-end' and data as 'back-end'. The business logic can reside in front-end, back-end or both depending on a range of factors beyond the scope of this syllabus.

These software layers are implemented in a tiered 'physical' architecture dependent on the targeted platform(s), such as:

- Web
- Desktop
- Mobile
- Server
- Cloud

In a single tier implementation, all software layers reside on one platform, such as a stand-alone desktop application. A 2-tier client-server approach sees two of the software layers reside on one platform either with a ‘thin client’ providing only the presentational layer, such as a legacy mainframe application, or ‘fat client’ with both presentation and business logic on a single platform accessing data on a separate server. Many web and mobile applications are implemented in 3-tiers, with separate or multiple servers used for business logic and data.

Software architecture – impacts on testing in the lifecycle

The context of the platform(s) for software development and deployment as well as the underlying software architecture will have an impact on the approach to software testing in the development lifecycle. Factors such as:

- The number of interfaces requiring integration testing, either at component, application or system level
- The deployment of software on multiple platforms and/or user interfaces such as web browsers
- The methods used for data access including the use of Application Programming Interfaces (APIs)

can influence the test approach for:

- system performance
- error handling and messaging
- network connection failures and restarts
- security, including penetration testing and encryption validation
- automation and other tooling
- test environment design

Having an awareness of the complexities of the underlying architecture, interfaces and data access methods can enable testers to add value to the technical levels and types of testing, for example:

- multi-tier architecture often means more non-functional testing is needed (e.g. for performance and reliability).
- building test environments that are suitable to meet testing objectives, which may initially necessitate the use of stubs and drivers.

This is especially the case in Agile ‘whole team’ development approaches but is also the case for more traditional sequential development lifecycles, helping to understand where to focus activities at each particular test level and type.

Lifecycles, roles and testing activities

The roles and corresponding activities of project development team members are more likely to be distinct in traditional sequential lifecycle models whereas in Agile development, it is more likely to find collaborative sharing of activities and multi-skilled roles.

However, there are many different ‘flavours’ of approach from ‘pure’ Agile to the implementation of some particular aspects of Agile coupled with sequential aspects in a more ‘hybrid’ approach. In the former approach, all traditional levels and types of testing could be performed in an iteration, whereas a more hybrid approach could see software developed in an iterative approach but with higher test levels and certain types of testing performed by a separate team in another testing environment. A range of different factors may dictate the exact lifecycle approach and it is important for a tester to recognise these and both adopt and adapt their testing activities accordingly – this is covered in more detail below under 'Choosing appropriate testing activities'.

Test automation often features strongly in successful Agile teams, noting some important differences to what this means for Unit Testing compared with Functional Testing at other test levels (see below). Test automation is a key enabler for the practice of DevOps with the automation integrated into the processes of Continuous Integration and Deployment of applications.

Testing skills and behaviours common across roles and activities

Collaboration between development team members is important regardless of lifecycle approach with Agile explicitly facilitating this with a 'whole team' responsibility for quality.

With the increasing focus on the automation of testing and the use of frameworks and tools, testers often need to acquire knowledge and skills to enable them to utilise tools and automation frameworks, whilst developers are more likely to be engaged in building and utilising the frameworks. The role of SDET (Software Development Engineer in Test) or similar is found in many organisations.

'Pairing' between developers and testers is also frequently found especially in Agile development, enabling the sharing of skills to help improve both the quality and throughput of their work, noting that pairing across a range of different roles and activities is also good practice for the same reasons. A good technical skill for a software tester to possess is the ability to scan and read programming code unit tests to understand the structure and identify potential areas that would benefit from testing. An important soft skill for a tester to possess is effective communication and the ability to adjust responses to different audiences.

Although the objectives, methods and tools for testing may differ between roles, test levels, test types and lifecycle approaches, there are several important aspects that share common features, including:

- Establishing test coverage to meet the test basis
- Designing tests to meet the desired test coverage
- The use of test techniques to support test analysis & design
- Repeatability of tests and replicability of defects
- Maintaining traceability between key testing and development artefacts

Although the formality, documentation and use of tools and techniques to support these activities may vary considerably, (notably the lighter approach to documentation and the more prevalent use of Exploratory testing in Agile), the tester must find a way that does not compromise on these key quality aspects.

Further detail on these can be found in the section on Test Analysis and Design and below, considering some of the benefits than can accrue from cross-skilling for all involved with testing activities.

Tester appreciation of software architecture and developer testing

There are many opportunities for testers to add value to the quality of a product by having an appreciation of, and an ability to contribute to the quality during Unit Testing and some of the more technical testing types and activities. Having an understanding of software architecture, development paradigms plus the tools and methods used by developers for testing is essential for achieving this added value.

Firstly, consider some aspects of developer testing. Developers typically use sophisticated Integrated Development Environments (IDE) and Unit Test Frameworks (e.g. xUnit, NUnit), to create specific testing code which is built to test the actual code, sometimes in a test first approach using Test Driven Development (TDD). In developer terminology, this method for Unit Testing equates to test automation. Some Unit Test Frameworks also integrate with automation testing tools and the building and use of automated test frameworks facilitates ‘continuous testing’ to support Continuous Integration, Continuous Deployment and DevOps practices. Contrast this with a legacy development platform where such Unit Test features often do not exist. Where automated Unit Testing is not performed, developers still need to manually test their code.

In addition, static and dynamic code coverage diagnostics often feature in a modern Integrated Development Environment (IDE) – contrast this with a legacy development platform where such features often do not exist. In any approach, it is important to define the coverage required and demonstrate that it is achieved.

Benefits of multi-skilled testers

Some substantial benefits can accrue from a tester possessing some of the technical knowledge and skills outlined in this topic area, including:

- Assist in the root cause analysis of defects
- Assist in the review of code quality
- Contribute to a variety of technical testing activities, including
    - The creation of code and scripts for automated testing and use of automated test frameworks
    - Testing for a variety of quality attributes, including Security & Performance efficiency (and section on Performance efficiency testing)
    - The use of tools to support test data creation, integration testing and result checking
- Awareness of the coverage of different levels and types of testing can help to avoid duplication of effort and pinpoint gaps in coverage against a test strategy and plan
- Assist developers in the use of techniques to improve the coverage for typical measures within Unit and Integration Testing
- Facilitate further cross-skills development and collaboration between testers, developers and other team members
- Drive improvements in communications across the development team
- Assist with testing in the support and maintenance stage
- Be able to give informed holistic recommendations if testing has a limited scope due to time or budget

In general, an understanding of software design patterns and development paradigms as well as software architecture will help testers to provide more value applying their testing skills when working closely with developers. For example, a tester having an understanding of object-oriented design is useful to help ensure that adequate test coverage is achieved when testing functions containing “private” data which may not be readily accessible.

Testing and Interfaces with Other Processes in the Lifecycle

Testing is relevant to both the software development life cycle (SDLC) and the whole product life cycle, since testing challenges will continue to arise after initial development has been completed.

Regardless of the development life cycle used, the testing process interfaces with other development processes such as project management, change and configuration management, software development and deployment and technical support. The nature of how these interfaces are planned and conducted in terms of people, processes and workflows will be influenced by, and take on the characteristics of, the life cycle in place. For example, with DevOps practices, development and operations processes are automated to continuously test, integrate and deploy software.

Choosing appropriate testing activities

Based on a foundation knowledge of the characteristics of testing in different life cycle models, the expectation from this syllabus is that a candidate will be able to analyse a situation, identify the nature of any life cycle(s) described within the situation, then recognise and address the associated testing challenges. Characteristics of iterative and sequential lifecycles that should be considered include:

- Situations that may include a 'hybrid' approach between iterative and sequential for a project or within an organisation
- The impact of the software architecture characteristics
- How levels and types of testing are incorporated into processes and workflows
- The detail and type of testing assets to support repeatability of tests and defects
- The formality of review types - e.g. in Agile, informal peer reviews and more likely to be prevalent than formal reviews due to time constraints plus the nature of regular and close collaboration within agile teams
- The application of experienced-based test techniques - In Agile, a higher percentage of test execution effort is dedicated to exploratory testing than in sequential projects - See the section on Test Analysis and Design for more detail
- The interaction between people and processes including the required soft skills (relevant in particular for effective collaboration in agile projects)
- The requirement for additional technical understanding and skills to complement testing skills
- The automation of processes and activities to support build & release management taking into account organisational constraints on release frequency.

In each case a variety of factors are expected to impact the way applications are tested. These factors include:

- Software development paradigm/method (e.g. Object-oriented design)
- Hardware and location
- The need for special test environments
- Audit/governance/regulatory requirements
- The applicability of specific test techniques
- Top-down versus bottom-up integration testing approaches
- Operational profile of use
- Software product quality attributes

Applications also fall into business domains related to the nature of the application itself, as in banking applications or image manipulation applications. Knowledge of the differences between such groups of applications is not required, except in so far as the application has attributes that make compliance with safety critical or business critical criteria an essential part of the testing. In such cases the implications of the level of criticality of the application on testing should be recognised and identified. This may include the imposition of Industry standards which prescribe the achievement of particular coverage levels and documentation requirements and/or the use risk-based approaches to manage and balance the perceived product and risks (and section on Testing and Risk).

2. Testing and Risk (12.5%)

Introduction to risk and risk-based testing

A risk involves is the possibility of an event in the future which has negative or undesirable consequences.

Risk management involves:

- Identifying new risks through use of historical information and knowledge and by involving the broadest possible sample of stakeholders
- Analysing identified risks to establish their level by determination of likelihood and impact
- Mitigating risks to reduce their likelihood of occurrence
- Developing contingency plans should the risks occur

A newly identified risk should be classified as either a project (a.k.a. planning) risk or a product (a.k.a. quality) risk. A product risk involves the possibility that a work product may fail to meet the needs of its users and/or stakeholders whereas a project risk impacts a project's ability to achieve its objectives, primarily to deliver on time and in budget.

Product and project risks may interact with each other. A project risk, if not effectively mitigated may lead to the introduction of a product risk. For example, a project risk whereby developers may have inadequate skills in the programming language to be used would, if not mitigated through training and/or recruitment, lead to the development of poor-quality software thus introducing a product risk. Similarly, a product risk, if not effectively mitigated may lead to the introduction of a project risk. For example, if a much larger than expected number of failures were to be detected in the software during functional accuracy system testing this would, if not mitigated through effective and timely debugging, introduce a project risk whereby testing and project schedules were adversely impacted due to extra confirmation and potentially regression test effort.

This risk interaction can also occur within a classification. For example, if the product owner on an agile team has insufficient skills in the development of well specified user stories (a project risk), this might lead to a further project risk by introducing difficulties and delays in the implementation of those user stories during sprints.

One risk-based testing technique, Failure Mode and Effect Analysis (FMEA), identifies how a component or system may fail, the effects resulting from each failure and the causes of each failure. In a similar manner, when analysing a risk, we must also consider whether its impact could lead to further risks. This continues until we cannot identify further related risks, i.e. we reach the end of a chain. FMEA is discussed in the ISTQB© Advanced Test Manager syllabus.

The distinction between these classifications is particularly important with respect to the how a risk is best mitigated. See section on Risk mitigation activities for product and project risks.

Risk analysis

Risk assessment is performed on all newly identified risks. It involves categorising each risk, establishing the risk level through determination of impact and likelihood, evaluating or assigning other properties, such as a risk owner. Risk assessment continues through the lifetime of a risk, with regular re-evaluation of the risk level as events occur that affect both likelihood and/or impact.

The likelihood of occurrence is influenced by many technical factors, including but not limited to, the complexity of the technology, budget and schedule constraints, personnel and training issues and late changes.

The impact upon occurrence is the severity of the effect on the users, customers, or other stakeholders and is therefore also known as business risk. There are also many business-related factors to consider, including, but not limited to, frequency of use of a feature, its criticality, loss of business, legal costs and costs to repair.

The level of risk can be assessed either quantitatively or qualitatively.

A qualitative risk level is often achieved by assigning values such as high, medium and low to both likelihood and/or impact factors. The risk is then assigned a level by using a risk matrix, where high impact/high likelihood risks would be assigned the highest level, with low high impact/low likelihood risks assigned the lowest level. Qualitative assessments of risk levels can be combined through multiplication or addition, to create an aggregate risk score For example with High being assigned the value of 3, Medium with 2 and Low with 1, a High Impact/Low Likelihood risk could calculate its level by multiplying the two factors together to reach a level of 3 whereas and a Medium Impact/High Likelihood risk would achieve a level of 6.

A quantitative risk level is often achieved by expressing the likelihood of occurrence as a percentage and the impact as a monetary cost which must reflect all business risk factors. Likelihood and impact are then multiplied together to achieve a monetary sum which reflects the risk level, with the highest monetary sum risks being acted upon first. The latter is more difficult and time consuming to perform but does get stakeholders to reflect on the true impact a risk can have on their business.

There are several lightweight and heavyweight risk-based testing techniques used for determining risk levels, qualitatively and quantitatively, which are discussed in the ISTQB$^{©}$ Advanced Test Manager syllabus.

If an organisation has a test policy document and/or test strategy document, then these should describe the general process by which product and project risks are managed, including the method for determining risk level.

Risk mitigation activities for product and project risks

When a product risk is associated with specific functional or non-functional quality characteristics of the test object, it is also called a quality risk. A quality risk can be mitigated by designing, implementing, and executing tests to cover the risk. As the tests pass, the likelihood of the risk occurring reduces, increasing stakeholders' confidence in the test object. The metrics from the executed tests and associated defects helps the stakeholders decide whether to extend testing or to transfer the remaining risk onto the users, customers, help desk/technical support, and/or operational staff.

The effort associated with developing and executing should relate to the level of risk it covers. The risk level influences which test techniques should be employed (See section on Selecting Test Techniques), the most appropriate test levels and test types to be performed and the extent of testing to be carried out. For example, an Agile project is to deliver, over a series of sprints, a new user interface for a critical feature used by call-centre staff when handling customer complaints. A quality risk has been recorded stating that the interface might not be intuitive and difficult to learn. There should then be emphasis on usability testing during the sprints. In another example, a government's taxation system requires a change to an algorithm which calculates the amount of tax relief due. A quality risk has been recorded stating that incorrect tax relief calculations would likely result in serious customer complaints. There should then be emphasis on functional accuracy testing during all test levels. To take one more example, suppose that a quality risk has been recorded stating that the proposed change to a file transfer mechanism between the host and external system might result in loss or corruption of data. There should be emphasis on performing system integration testing in a live-like environment.

The effort/cost of risk mitigation may also be spread across several related risks. For example, the costs of setting up performance tests (tools, environments etc) may be spread across several product risks which potentially could be in different projects. The link between risk mitigation and effort should therefore also be considered at organisational level and mentioned in a test strategy (where one exists).

Project risks cannot directly be mitigated through test design and execution. Instead they are mitigated through other project activities. Project risks that directly impact testing may or may not be solely within the power of the test team to mitigate. For example, if a project risk stated that test environments not built and configured correctly would be likely cause delays in the testing schedule then a mitigating action could be to perform smoke testing on the environments before full deployment. If, for example, a project risk stated that the developers on an agile project had limited testing skills which might result in a high defect yield during system testing, then a mitigating action might be to pair developers with testers during unit and integration testing to facilitate a skills transfer. One final example, if the user stories for an agile project were found to lack the information needed to support accurate estimation during the sprints planning poker sessions, then a mitigating action might be to introduce more formal reviews of the stories before they were placed in the product backlog.

Project risks could indirectly be mitigated through test design and execution. For example, the application of the most appropriate test design techniques, resulting in an effective minimal set of test cases that detects defects early, thus positively impacting project timescales.

3. Test Management (17.5%)

Test Planning

This topic builds on the knowledge from the ISTQB Foundation Certificate covering the hierarchy of test management documentation, including test policy, test strategy, project test plan and level test plan. Information and sample test plans can be found in ISO/IEC/IEEE 29119-3.

Candidates will be expected to analyse document hierarchies to determine whether policy, strategy and planning issues are adequately documented, based on the nature of the project, including risks and system development life cycle (SDLC) employed, noting that the emphasis is on the planning activity and documentation rather than who is taking the role of test manager. Candidates should be able to identify subordinate plans for:

- Test levels
- Test types
- Releases
- Iterations

Selecting and using test entry and exit criteria

Building on the knowledge from the ISTQB Foundation Certificate for Test Objectives plus Entry and Exit Criteria for different levels of testing (noting that more typically these are referred to as ‘Definition of Ready’ and ‘Definition of Done’ respectively in Agile Development) candidates are expected to be able to analyse testing situations and select appropriate test entry and exit criteria to meet defined objectives.

They will also be expected explain what alternatives are available when they are not met. Key situations for which entry criteria are not met are:

- Specified software quality is not met, in which case a decision needs to be made whether to accept lower quality and revise aspects of the test plan and/or defer to the next iteration
- Key resource or infrastructure not available, in which case a decision needs to be made on whether to wait or acquire additional resources and whether to revise aspects of the plan to cater for the unavailability

Key situations for which exit criteria are not met are:

- Coverage targets are not met, including quality characteristics
- The number and/or severity of unresolved defects is above the planned acceptable level
- Test budget has been/will be exceeded or schedule will not be met

In all cases a decision needs to be made on whether to relax the criteria and accept residual risk, defer to another iteration, or acquire more time/resource and/or find productivity improvements. The decision must take into account the organisation/project approach to risk management and test policy (and section on Testing and Risk)

Improving a defect management process

Building on the knowledge from the ISTQB Foundation Certificate for defects and defect management, improvements to a simple defect management process should consider the following typical factors:

- Criteria to meet specified objectives for process improvement (development and/or testing)
- Workflow deficiencies
- Information traceability
- Defect repeatability
- Communication

Information about types of defects and reviews can be found in ISO/IEC 20246 – Work Product Reviews, and a sample defect report can be found in ISO/IEC/IEEE 29119-3.

4. Test Analysis and Design (20%)

The benefits and pitfalls of deploying test techniques

Test techniques are applied during test analysis and design to derive test conditions, test cases, and test data from the test basis. There are three categories of test techniques, black-box, white-box and experience-based.

Black Box techniques

Black-box techniques are applied to elements of the test basis typically including requirements, specifications, use cases, and user stories. Using the test basis as a source of information, the techniques enable specific aspects to be modelled from which tests can be systematically derived. For example, a user story may contain information about a how a user navigates their way through an application. A model can be constructed from this information in the form of a state transition diagram. Using the state transition testing technique, test cases are designed from the model which enable a particular model coverage to be achieved. For example, test cases to enable 1-switch coverage of valid transitions may be designed. Advantages of applying black box techniques include:

- They identify tests that might otherwise be missed through manual analysis of the test basis, for example not considering all combinations of triggering input conditions in a decision table
- They provide a measurable level of coverage based on the items tested in the test basis. For example, with boundary value analysis, measuring the percentage of all 3-point boundary values tested for an ordered equivalence partition model
- They help address one of the seven testing principles, exhaustive testing is impossible, by creating a more effective set of tests for defect identification in the test object For example, combinatorial techniques such as pairwise can be used to execute all possible discrete combinations of each pair of input parameters on a user interface with the smallest set of test cases, as industry experience tells us that most defects are caused by interactions of, at most, two interacting parameters. The application of the pairwise technique is discussed in the ISTQB© Advanced Test Analyst syllabus
- They can help identify defects in the test basis early in the project lifecycle. For example, the inability to apply boundary value analysis because minimum and maximum partition values cannot be determined from the specification of a user story. If attempting to apply BVA as part of a review, such defects can be detected before dynamic testing is conducted.

The disadvantages of applying black box techniques include:

- They require a well specified test basis
- They require testers with adequate skills in the application of the techniques
- By requiring that the test basis be analysed, they take time to apply compared to ad-hoc testing, especially if not using available tool support. Techniques, such as pairwise, have several tools available.

White Box techniques

White-box techniques are applied to elements of the test basis typically including architecture, detailed design, internal structure, or the code of the test object. Unlike black-box test techniques, white-box test techniques concentrate on the structure and processing within the test object. They share a similar set of advantages and disadvantages as black-box techniques. Advantages include:

- They can identify defects in the test basis that are not easily detected by black-box techniques. For example, within code structures, broken or incomplete paths, logic defects in conditional loops, missing ELSE statements
- They identify tests that might otherwise be missed through manual analysis of the test basis, for example not considering the paths through a code structure to achieve a desired level of path coverage

- They provide a measurable level of coverage based on the items tested in the test basis. For example, with decision testing, measuring the percentage of all decision outcomes tested in a code component's structure

Disadvantages of white-box techniques include:

- They require developers/testers with adequate skills in the application of the techniques, which can be complex to apply
- They take time to apply and require tool support, except for trivial code. Accurate structural coverage measurement requires the use of tools.

White-box techniques are discussed in the ISTQB© Advanced Technical Test Analyst syllabus.

Experience-based techniques

Experienced-based techniques include:

- error guessing
- exploratory testing
- defect-based
- checklist-based testing.

These techniques derive test conditions, test cases, and test data from a test basis that may include knowledge and experience of testers, developers, users and other stakeholders, defect taxonomies and checklists. Most of these techniques are regarded as un-scripted, i.e. they do not follow a documented test script, instead relying on a more lightweight test documentation approach for test execution.

The main advantages of experienced-based techniques include:

- They are effective at finding defects that are not easily found by black-box techniques, such as scenario-based and workflow-related defects missed during scripted testing (where test execution is carried out by following a previously documented sequence of tests), defects that fall between functional boundaries, and some performance and security defects.
- They may be a good alternative to more scripted testing (which may employ black-box techniques) in cases where system documentation is poor or when testing time is severely restricted
- They help the team become familiar with the software as it is produced
- They can leverage the expertise in the domain and technology of those not directly involved in testing (e.g. product owners in agile development)

Exploratory testing is especially useful in agile projects due to:

- Limited time for test analysis and design imposed by iterations/sprints
- Greater experience and knowledge gained through agile team collaboration
- A higher level of automation across all test levels allowing more focus on manual test execution of new features.

The main disadvantages of experienced-based techniques include:

- They have limited defect finding capacity when not performed by experienced, knowledgeable and intuitive individuals
- They do not produce reusable test procedures. Test cases/procedures may have to retrospectively created where defects are found (reflecting gaps in scripted testing)

- It can be difficult to track the tests in a test management system. With session-based-test test management (SBTM), test cases can be added in test management tools that are correspond to the exploratory sessions
- Failures detected during test execution may prove difficult to debug without a record of the steps taken before the failure occurred. Some organisations get around this problem by using automation tools to record the steps performed in test execution
- The ability to precisely assess test coverage is limited.

The best test coverage is often achieved with a blend of scripted testing using black and white-box techniques where appropriate, augmented by experience-based techniques.

Selecting test techniques

There are many factors that influence which test techniques should be applied during test analysis and design. These can be broadly grouped as follows:

Organisational

An organisation’s test policy and test strategy may define rules for applying test techniques. For example, 2-point Boundary Value Analysis, rather than Equivalence Partitioning, shall be applied as a minimum level of coverage on all ordered partition models. For example, Decision Testing shall be used as the minimum level of structural code coverage. Some organisations may define a risk-based rule system in the test strategy, where the choice of technique is linked to the level of risk. For example, Statement Testing for code components with the lowest risk level (i.e. low likelihood and low impact), Decision Testing for medium level risk components and Multiple Condition Testing for all high-level risk components. Such rules should also be applied in the project test plans.

Regulatory/Standard compliant

Sometimes the coverage metric required for test techniques may be derived from applicable standards that apply to the type of system under test. For example, in the aeronautics industry, it may be required to conform to standard DO-178C (in Europe, ED-12C.) This standard contains the following five failure conditions:

A. Catastrophic: failure may cause lack of critical function needed to safely fly or land the plane
B. Hazardous: failure may have a large negative impact on safety or performance efficiency
C. Major: failure is significant, but less serious than A or B
D. Minor: failure is noticeable, but with less impact than C
E. No effect: failure has no impact on safety

Each of the first three levels, A to C, must be tested by applying a specific white-box technique as the minimum level of coverage. For example, Modified Condition/Decision Coverage for level A. Project test plans must clearly define these regulatory/standard compliant rules.

Risk

In a risk-based testing approach, the risk level will principally determine the depth of testing to be performed using a given test technique. For example, in state transition testing 1-switch coverage might be used for lower risk levels, 2-switch coverage and above for higher risk levels. Equivalence partitioning might be employed to test the positive partitions for lower risk levels whereas BVA employed to test the negative partitions for higher risk levels. Furthermore, 2-point BVA might be employed for lower risk levels and 3-point BVA for higher risk levels.

Constraints

Constraints such as limited time or budget influence the degree to which experience-based test techniques should be employed. As discussed in the section on Experience-based techniques, a higher percentage of the test effort is often expended conducting exploratory testing in agile projects than in sequential projects and indeed this may be the only viable technique to apply where time constraints are severe.

Available test basis documentation is also a factor. Black-box techniques require a well-specified test basis. If requirements, for example, are poorly specified or not at all, which can occur in some legacy systems, then exploratory testing may be favoured instead of scripted testing.

Experience

As discussed in the section on Experience-based techniques, most experienced based techniques require knowledgeable, experienced and intuitive testers for effective test design and execution. Defect-based techniques, however, can be applied by less experienced testers as the information on likely defects already exists within a defect taxonomy.

Analysis of the test basis

Most test techniques are chosen during test analysis and design, when the tester is analysing the test basis (requirement, user story, use case, code etc) and designing the tests. For example, if the requirement contained complex business rules from which triggering conditions and resulting actions can be determined, then decision table testing would be an obvious choice. If a code structure contained decision outcomes, e.g. from IF or CASE statements, then decision testing would be the best choice for minimum structural coverage (note that if such statements contained multiple conditions then other white-box techniques discussed in the ISTQB© Advanced Technical Test Analyst syllabus might be more appropriate).

The best test design is often achieved by applying more than one technique. For example, a test case for each rule in a decision table with further tests applying 2-point boundary value analysis to numeric conditions in the table.

Information traceability and test coverage measures

Building on the concept of information traceability between work products from the test process plus the different types of test coverage measures that are part of this traceability, this topic considers the factors that may impact the effectiveness of the test coverage measures and information traceability required.

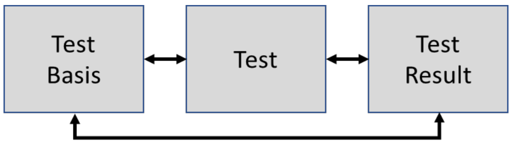
Test coverage measures, whether applied to functional, non-functional or structural aspects, are key criteria in providing management information as well as helping to drive the test process. They define ‘what’ needs to be tested in order that the test basis can be considered as adequately met. Building on the concept of coverage criteria acting as key performance indicators (KPIs) it is important to be able to track testing work products from the test process back to the coverage criteria such that accurate management information can be provided to stakeholders on the status of the required coverage against the test basis.

The ISTQB Advanced Test Manager syllabus considers in some detail the factors that may determine the type and level at which to specify test coverage measures and targets in the form of features and test conditions. This syllabus considers the traceability model involving these measures and targets based on the types and uses of test coverage measures already established in the ISTQB Foundation Syllabus.

Models for information traceability

Firstly, consider a simple model that relates testing to the test basis in a generic fashion:

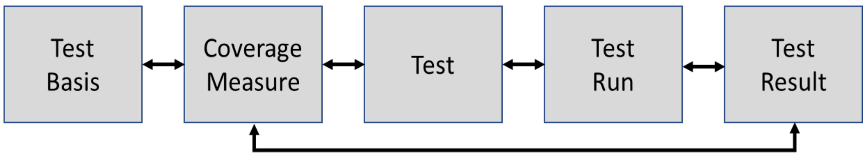
Model A



This model simply demonstrates that results from running tests should be traceable back to the basis of those tests.

Now consider a model where some typical measures are specified to demonstrate that the test basis is adequately covered by testing. Also, take into account that tests may be run a number of times and that it is useful to be able to determine the coverage met for each run.

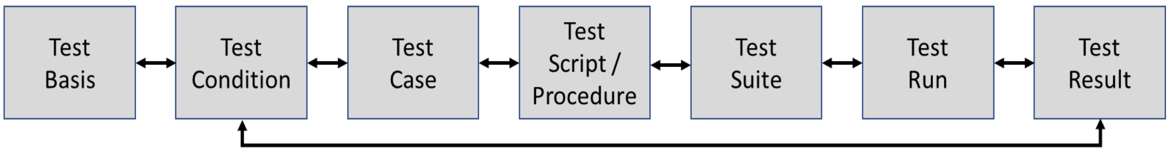
Model B



Note that in this model, test results are traced to the coverage criteria which in turn demonstrates that the test basis is met.

Finally, consider a generic model of testing work products and the requirement for bi-directional traceability established in the ISTQB Foundation Test Process:

Model C



Comparing this with the simpler models A and B means:

- A coverage measure can be considered as equivalent to a test condition: (i.e., a work product from the test analysis activity which may be organised by feature to be tested)
- A test can be considered as equivalent to the work products test case, test script/procedure and test suite (i.e., all typical work products from test design and test implementation)

Note that these models are not designed to define any strict data model of the work products. There could be a variety of ways in which these and any other work products may be organised but in general, the bi-directional arrows highlight the fact that complex many-to-many relationships could exist between the work products.

A range of factors determines what the actual work products (e.g. test strategy, test plan, test type, test level) would look like to support this model. Principal factors include the level of formality (formal to informal) and development life cycle model being followed. The need for formality and the ability to trace any work products that support these models also varies considerably according the following factors and considerations:

- Size and complexity of software and systems under test
- Product and project risks
- Internal and external stakeholder requirements/expectations
    - Test monitoring and control
    - Auditability
- Level of automation involved in the test process
- Change frequency and the ability to measure the impact of change on testing
- Regression test requirements

A key requirement in this generic model for traceability is that the results of testing need to be traceable to any test coverage measure used, which in turn can be traced to the original test basis from which the test coverage measures are derived. Remember, it is the test coverage measures that define 'what' needs to be tested in order that the test basis can be considered as adequately met.

Typical scenarios for test coverage measures and information traceability

The definition of test coverage measures is an imprecise science, since the level at which they are defined is contextual within a range of typical factors as outlined above. Some measures can be more formally derived as part of test analysis and test design from the use of test techniques, but others are from less formal analytical decomposition based on one of more of those factors.

The following lists some generic factors that affect the level of detail for specifying test coverage measures taking into account requirements for information traceability:

- Less likely need for detailed test coverage measures when:
    - Software and systems are smaller or less complex (but see below re iterations)
    - Lower product and project risk scenarios
    - The test basis is already defined at a detailed level (and therefore making it easier to relate specific tests to the test basis directly)
    - Technology facing testing:
        - Testing is 'self-contained' either within the context of white box testing or simple integration of software and systems
        - Testing of non-functional software product quality characteristics

- Likely need for more detailed test coverage measures when:
    - Software and systems are larger and/or complex
    - Higher product and project risk scenarios
    - The test basis is defined at a high level (and therefore making it more difficult to relate specific tests to a specific aspect of the test basis directly)
    - Business facing functional testing
    - The need for testing to be auditable
    - Less formal test design documentation is in place (as per Model B)

For example, if looking at code coverage from testing, this would typically be analysed by tools and take into account specific aspects of code coverage which can be established using a range of white box test techniques. This is further covered in the ISTQB Technical Test Analyst syllabus, but suffice to say that it is not typical to define a complex model for information traceability where the structure of the test basis is effectively self-contained, potentially self-documented within the code and supported by a variety of analytical tools readily available and also where tests are frequently automated.

Taking into account more generally the technology facing aspects of testing, it is less likely that detailed test coverage measures are defined for unit testing and the testing of most non-functional software product quality characteristics such as those categorised in ISO standard (ISO/IEC 25010) – see section on Software Product Quality Characteristics. Conversely, business facing functional testing is more likely to require the definition of more detailed test coverage measures for testing in terms of features/sub-features and specific detailed conditions to be met.

There may be several factors that conflict when considering the level at which to define test coverage measures and the information traceability required. This is typically the case when using an approach (Agile or otherwise) that does not call for detailed documentation and more extensive use of Exploratory Testing. For example, there could be:

- External regulatory requirements to meet which dictate a level of auditability of testing requiring detailed proof that the test basis has been adequately covered, for example:
    - Industry specific requirements, notably the financial sector
    - Government and legal requirements such as GDPR (General Data Protection Regulations
    - Safety critical requirements
    - Demonstrating adequate coverage of negative (alternative path/sad path) testing as well as positive (main path/happy path) testing
- Although an iteration in Agile may be small and relatively self-contained, it could be part of a much larger/complex release or system
- The software or system is undergoing frequent change and it is important to be able to analyse the impact of that change on any regression tests to ensure that they are still effective

An important point to note is that Model B does not imply less detailed test coverage measures should be in place – in fact, quite the reverse. The level of formality in test design is a key factor in the level at which test measures should be defined in terms of maintaining information traceability. In general, the less formality/detail in the test design, the greater the importance of establishing detailed test coverage measures. Other factors may conflict and dictate otherwise as outlined above e.g. detailed requirements exist, low risk tests etc. Additionally, whilst it may appear in Agile projects that test coverage is established by the definition of Acceptance Criteria against the test basis of User Stories, it may well be necessary to define more detailed test coverage measures to provide adequate information traceability according to different circumstances.

Actions to mitigate the risk of inadequate information traceability

In general, it is important to agree the level of test coverage measures with internal stakeholders to ensure that their expectation in terms of any reporting requirements and ability to effectively monitor and control testing are satisfied.

Additionally, although the use of automation assists repeatability and auditability of testing, it is still important to be able to understand what coverage is being achieved and the impact of any changes on the effectiveness of existing tests, as well as being able to repeat the testing.

Some tools help provide traceability between various test assets and defects. In general, even where test assets may not be documented in any detail, it can be useful to create an overarching framework of test coverage especially in the event of one of the conflicting scenarios outlined above.

5. Software Product Quality Characteristics (22.5%)

Introduction to testing software product characteristics

Considering software product quality characteristics, such as those categorised in ISO standard (ISO/IEC 25010), helps the tester to avoid missing important risks that may have a severe impact on a product's success if testing approaches are not adopted to mitigate them. The following factors influence the general risk of failing to recognise issues relating to product quality characteristics:

- Requirements often focus on functionality and may not specifically mention other aspects, such as performance and security. This is a particularly common situation where the requirements are defined by Product Owners. They may simply consider such characteristics as being "given".
- There may be a fundamental lack of awareness for different product quality characteristics within the project. Again, this may result in a singular focus on functionality.
- Implementing testing approaches for non-functional aspects such as performance efficiency often places demands on resources (e.g. environments, tools) which a project cannot easily meet. As a result, the "inconvenient" risks tend to become overlooked.

Testers cannot be expected to test for all product quality characteristics, but they do need an awareness of typical risks and appropriate testing approaches, especially for the characteristics which commonly apply to a wide range of products. These include:

- Security
- Performance efficiency
- Usability
- Portability

Security testing

Typical security risks

Security risks arise from the loss of confidentiality, integrity, or availability of information or information systems.

A security risk assessment focusses on the following aspects:

- Identifying threat sources (e.g. hostile cyber or physical attacks, human error)
- Identifying any malevolent actions that could be produced by those sources
- Identifying the vulnerabilities within an application that could be exploited by threat sources through their malevolent actions

Typical types of security vulnerabilities include:

- Unauthorised copying of applications or data
- Unauthorised access control which circumvent user login mechanisms
- Insertion of malicious code into a web page that may be exercised by subsequent users (cross-site scripting).

- Entering data strings into a user interface input field that are longer than the program code can correctly handle (buffer overflow). This represents an opportunity for running malicious code instructions.
- Denial of service, which prevents users from interacting with an application (e.g. by overloading a web server with “nuisance” requests)
- The interception, mimicking and/or altering and subsequent relaying of communications (e.g. credit card transactions) by a third party such that a user remains unaware of that third party’s presence (“Man in the Middle” attack)
- Breaking the encryption codes used to protect sensitive data
- Maliciously inserted code and which activates only under certain conditions (e.g. on a specific date) and may perform malicious acts such as the deletion of files.

Applications may be vulnerable to security threats if any of the following applies:

- The application uses the internet. This is one of the primary enablers for threat sources to expose security vulnerabilities.
- User accounts and access rights are not managed properly, resulting in inadvertent or intentional abuse of user privileges.
- Certain data is used or created which is classified as “sensitive” and must be secured. The classification may be determined by the organisation’s data policy or governed by law.
- Configuration management is not practiced. The risk is that an unauthorised change to a secure device could cause a security vulnerability that might go undetected.
- Mobile devices are used. These have a unique set of security concerns. These devices have a high risk of contact with malware.
- Secure password practices are not applied.
- Malware protection is not practiced, so that malware can be prevented, detected and removed.

Approaches to security testing

Security testing approaches consists of a mix of static analysis and dynamic testing.

Static analysis involves the following tasks:

- Code reviews using security-specific checklists. These are effective, for example, in detecting maliciously inserted code which activates only under certain conditions and is therefore difficult to detect with dynamic testing.
- Code analysis using tools These help the developer to identify violations of security-relevant coding principles before the code is used in a build.

Dynamic testing involves the following tasks:

- Examining dynamic behaviour in response to inputs that are designed to “attack” the application. Emphasis is placed on the ability of the application to handle unexpected conditions and in causing the application to crash (an application which has crashed invariably offers a potential hacker with sufficient information to insert malicious code). Tools may be used to simulate particular conditions that are difficult to create, such as interrupts from the operating system (e.g. hardware failures). Creativity is required in designing tests to uncover security weaknesses; testers are typically looking for the unusual and approach testing by adopting an attacker perspective. This is often known as ethical hacking or penetration (pen) testing.
- All application interfaces are considered as potential targets for attack vectors. This includes the user interface, file-based interfaces, and calls to external applications or services (e.g. using API calls).

- System testing is typically considered in conjunction with functional requirements. For instance, “when doing x the system should not allow y to happen.” As functional tests are conducted the tester should be probing for ways that security constraints might be violated.

Any security testing approach requires careful planning. A realistic operational environment should be used for system and acceptance testing so that it responds exactly the same to attacks as would be expected from the real application in day-to-day use.

Performance efficiency testing

Typical performance efficiency risks

The risks associated with performance efficiency break down into the following principal categories:

- Ability of an application to respond to requests within the required time. These requests typically relate to the user (e.g. response time for receiving a request for data)
- Ability of an application to use resources (e.g. memory) efficiently.
- Ability of an application to function within capacity limits, such as the maximum amount of data that can be transferred across a network connection (bandwidth) or the number of users that can use the system at any one time.

Any one of these risks may impact the system or application. When a system or application is placed under particular loads (see next section on approaches to performance efficiency testing), the following may occur:

- The application becomes intolerably slow to respond. If the response is to a user input, the user will become increasingly frustrated and, where alternatives are available, they may even abandon the application and seek out an alternative. In the case of web applications, the time at which a site is abandoned is typically seven seconds.
- The system may fail. This may happen if a particular resource exceeds its available limits (e.g. the main memory available to a real-time embedded application are exhausted).

To summarise, any application which works with limited resources or must respond within a particular time is exposed to one or more of the above-mentioned risks. The tester must respond to these risky by proposing a performance testing approach.

Approaches to performance efficiency testing

A critical aspect of any performance efficiency test is to specify the load conditions under which the application must perform. The parameters which are considered in a load specification are:

- The interactions which a user has with the application. These may be categorising according the different types of use (personas) who use the system (e.g. a person browsing a range of items on sale or a person making an order).
- Number of users performing those interactions at particular times (e.g. on a daily or yearly cycle).

Combining these two parameters enable the performance tester to define typical usage patterns for individual types of user (user profiles).

Any performance efficiency approach must analyse data and refine requirements in order to define one or more of the following types of load:

- Normal (typical) loads
- Maximum (peak) loads
- Stress (abnormal) loads

The definition and generation of the required loads are key aspects in a performance test approach. The tester specifies a particular load by defining the numbers of users performing particular user profiles. For example, a normal load may be defined which consists of 100 people applying the “browsing” user profile and 20 people applying the “buying” user profile. A stress load may be defined where 200 people are buying (typically the number is set to a higher number than the maximum permitted).

Generation of the loads can be achieved using a manual or an automatic approach:

- A manual approach involves gathering the required number of people together and asking them to perform the steps in a particular user profile. This may be a good approach for small numbers of users but suffers from a lack of repeatability and precision. Where large numbers of physical users are required, a “crowd testing” service may be applied.
- An automatic approach uses tools to simulate the required number of users (virtual users) to generate the required loads. The tools ensure good test repeatability but require effort to set up and validate the testing scripts to be executed by the tool. Load generation using tools requires computing capacity and infrastructure, in particular where large numbers of virtual users are to be generated. An important decision in the test approach is whether to purchase the test infrastructure or to use a cloud-based service.

When specifying a performance approach, the goal of the test must always be in the foreground. On occasions the goal set for performance testing may be more investigative in nature than is typical with traditional tests based on “pass/fail” criteria

Usability evaluation

Typical usability risks

Usability in general considers the following three aspects:

- Usability: How well and how effectively the user interacts with the application
- User experience: User satisfaction with the services obtained and experience of using the application.
- Accessibility: Degree to which the application can be used by people with disabilities.

Typical risks relating to usability include:

- Users return the application because it is in ineffective; they find they can’t do their daily tasks.
- Users realise they are more efficient at performing their tasks with a competitor’s application and decide not to purchase further licenses.
- The application works as specified, but users can’t figure out how to use it. They repeatedly need to call support and become dissatisfied and frustrated with the application.
- An application which has an unprofessional and chaotic user interface increases the risk of financial loss caused by damage to the producer’s reputation.

Typical user experience risks include:

- Users are unable to buy a product because the app they used to buy the product is complicated and deceptive to use. The potential customer abandons their shopping cart.
- The software product is usable, but the user documentation cannot be understood in their language and there is poor help desk response.
- The product arrives in an unattractive or impractical packaging
- The user does not enjoy playing the latest version of a game app and makes negative comments in their social media. The game develops a bad reputation and is rejected by the gaming community.

Typical accessibility risks include:

- The application cannot be used by people with sight disabilities because the font size used on the user interface is too small and cannot be enlarged.
- The user interface violates mandatory regulations and results in a financial penalty.
- The software product is not compatible with other software or hardware used by people with disabilities (e.g. voice input software cannot be used).

Approaches to usability evaluation

The following key points must be appreciated by the tester when specifying an approach to usability evaluation:

- An application can work exactly to specification and still have serious usability problems.
- With the exception of compliance checks conducted with usability standards, many of the usability tests performed detect issues which need further evaluation rather than defects.
- The tester is not the person who performs the usability test execution. Representative users perform these tasks under the instruction of the tester.

An approach to usability evaluation describes the activities required to perform usability testing, user experience evaluation and accessibility evaluation.

Reviews, survey and testing may be performed when evaluating different aspects of usability:

- Usability reviews enable stakeholders to evaluate the user interface of a software product for usability problems; the evaluation is based on their experience and may be supported by checklists.
- User surveys involve users answering questionnaires regarding their satisfaction with the software product. These questionnaires may be custom-developed for a particular project or use pre-defined industry-standards. Using a standardised questionnaire makes benchmarking against other products possible if required. Surveys are particularly useful for evaluating user experience and can provide a degree of measurability in a subject that is typically subjective in nature.
- Usability testing involves observing users while they perform typical tasks with the software product. The testing sessions may be conducted in an informal environment or in a fully equipped usability test lab. The principal requirement for usability testing is that the representative user is able to give feedback on their impressions and experiences whilst performing given tasks. In an informal usability testing session, the user feedback is recoded as notes, whereas a usability test lab allows unobtrusive observations and recordings to be made.

Portability testing

Portability characteristics describe the ability of the software to be installed, adapted or replaced in its intended environment.

The following risks apply to installability:

- Software and/or written procedures used to install the software on its target environment are incomplete, incorrect or not understandable. This may have the following impacts:
  - the installation fails to be completed (e.g. where available main memory is exhausted before the installation has been completed)
  - the installation is completed, but selected installation options and parameterizations used to configure the software have not been applied correctly (e.g. maximum number of users permitted).

    - the user inadvertently installs software which is not needed or omits features that they wanted (e.g. selection of a full installation instead or a partial or “basic” installation).
- The installation process is not completed within a particular period of time. This may be a critical parameter in the recovery of a failed system.
- The installation procedures fail to recognise or provide information about hardware or software compatibilities that are required (e.g. a particular operating system version required).

The following risks apply to adaptability:
- Software cannot be used if aspects of its environment (hardware or software) are modified. For example, software features may not function correctly if a new version of an operating system is installed.
- The software cannot be ported to various specified environments by performing a predefined procedure. The impact of these restrictions depends largely on whether the specified environments must be supported or can be considered as optional.

The following risks apply to replaceability:
- Other software products or software components cannot be used in place of the specified software product for the same purpose in the same environment (e.g. substituting one database management system to another). This may limit the flexibility of the product to be used by a large and varied user base.
- A commercial product cannot evolve and upgrade to make use of new technologies in hardware or software. This results in a reduction in the ability to market that product successfully.

6. Test Automation Characteristics (7.5%)

Test automation in the Software Development Lifecycle

In general, test automation supports the software development lifecycle by providing the following benefits:

- Test automation can be implemented early in the SDLC and thereby supports the principle of testing early in lifecycle. Concepts such as test-driven development (TDD) rely on the availability of automated tests.
- Testing spends less time on critical path during development. Test automation can be executed much faster than manual testing and enables unattended testing at times when manual testing is typically not performed. This can be particularly beneficial for projects using an Agile approach where the time available for testing is limited.
- Efficiency gains in testing enable projects to allocate available resources for other SDLC activities, such as better program design and hence lower cost of code maintenance.
- Test automation is an essential element of lifecycles which apply DevOps principles where testing is highly integrated into the lifecycle.
- The maintenance stage of a product’s lifecycle is supported by automated regression testing. This enables scheduled changes and emergency defect fixes to be tested quickly before deployment.
- Monitoring activities during the production stage of an application are supported by automated tests which regularly check for the availability of required services and for the overall availability of business-critical processes.

When considering the testing stage of the SDLC, test automation supports the individual test levels as follows:

- Component testing is supported by enabling testers and developers to implement automated unit tests. Frameworks are widely used to support these activities and are specific to the programming language (e.g. JUnit framework for Java).
- Integration testing is supported by implementing automated tests which focus on interfaces between component and systems. Where external interfaces are used (e.g. web services), their correct integration with the application must be checked regularly to ensure that interfaces have not changed unexpectedly. Testing of interface specifications is a routine task that can easily be automated.
- System testing is supported by automation for the testing of both functional and non-functional quality characteristics. Testing of characteristics such as performance efficiency would be both inefficient and ineffective without the use of test automation. Testing of system functionality is considered one of the principal areas where test execution tools are applied. Various test automation approaches, such as the data-driven approach, are used to ensure a modular and efficient implementation of the automated system tests.
- Acceptance testing typically makes use of automated tests by re-using those system tests which help focus on business processes. Note that investing in test automation exclusively for acceptance testing must be well considered to ensure a positive cost/benefit outcome.

Requirements of tests for test automation

Not all tests are suited to test automation and are best executed manually. Tests should undergo a selection process before deciding on whether to automate them. The following criteria should be considered in that selection process:

- Frequency of use. Any test automation involves effort for implementation and maintaining the automated scripts required. It makes no sense to automate a test which is not expected to be executed frequently.
- Time required for manual test execution. One of the benefits of test execution is to reduce the time required for test execution. If a particular test takes a long time to execute manually, it will contribute more to this objective and is therefore a good candidate for automation.
- Complexity of test. A test which is complex to perform manually has a higher risk of incorrect execution than a simple test. Automating the test will reduce this risk.
- Test dependency. An individual test may depend on the execution of another test as a pre-condition. Tests which serve as pre-conditions for other automated tests are good candidates for automation
- Test details: Test cases which provide details of steps to take and data to use will require less effort to implement.

At a project level, the decision to automate testing must also consider the costs of designing a test automation architecture and implementing a test automation infrastructure. Even if individual tests meet the criteria mentioned above, the overall cost of test automation may exceed the expected benefits. Note that when deciding to use a software development approach such as Agile, the cost of test automation must always be factored into the decision. For Agile projects the use of test automation is indispensable.