```
====== Appendix G:  Tests ======

[[dido:public:ra | Return to Reference Architecture (RA)]] or
[[dido:public:ra:xapend | Return to Appendices]]

^ Test Name  ^ Description ^
| [[dido:public:ra:xapend:xapend.a_glossary:a:acceptancetesting]]  |
<WRAP>**Acceptance Testing** is a testing technique performed to determine
whether or not the software system has met the
[[dido:public:ra:xapend:xapend.a_glossary:r:requirement|requirement]]
specifications. The main purpose of this test is to evaluate the system's
compliance with the business requirements and verify if it is has met the
required criteria for delivery to end users. \\
 \\
There are various forms of acceptance testing:\\

  : • User acceptance Testing \\
  : • Business acceptance Testing \\
  : • Alpha Testing \\
  : • Beta Testing \\
 \\
Source:
[[https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testin
g.htm]]</WRAP> |
| [[dido:public:ra:xapend:xapend.a_glossary:b:blackboxtesting]]    |
<WRAP>**Black Box Testing** is a type of software testing in which the
functionality of the software is not known. The testing is done without the
internal knowledge of the products. \\
 \\
Black box testing can be done in following ways:\\

    : 1. Syntax Driven Testing – This type of testing is applied to systems
that can be syntactically represented by some language. For example-
compilers,language that can be represented by context free grammar. In this,
the test cases are generated so that each grammar rule is used at least
once.\\
 \\
    : 2. Equivalence partitioning – It is often seen that many type of inputs
work similarly so instead of giving all of them separately we can group them
together and test only one input of each group. The idea is to
[[dido:public:ra:xapend:xapend.a_glossary:p:partition|partition]] the input
domain of the system into a number of equivalence classes such that each
member of class works in a similar way, i.e., if a test case in one class
results in some error, other members of class would also result into same
error. \\
 \\
Source:
```

```
[[https://www.geeksforgeeks.org/software-engineering-black-box-testing/]]
</WRAP> |
| [[dido:public:ra:xapend:xapend.a_glossary:e:end2endtest]]        |
<WRAP>**End-to-End Testing (E2E testing)** refers to a software testing method
that involves testing an application’s workflow from beginning to end. This
method basically aims to replicate real user scenarios so that the system can
be validated for integration and
[[dido:public:ra:xapend:xapend.a_glossary:d:dataintegrity]]. \\
 \\
Essentially, the test goes through every operation the application can perform
to test how the application communicates with hardware, network connectivity,
external dependencies, databases, and other applications. Usually, E2E testing
is executed after functional and system testing is complete. \\
 \\
Source: [[https://www.browserstack.com/guide/end-to-end-testing]]
</WRAP> |
| [[dido:public:ra:xapend:xapend.a_glossary:i:integrationtesting]] |
<WRAP>**Integration Testing** is performed to test individual components to
check how they function together. In other words, it is performed to test the
modules which are working fine individually and do not show bugs when
integrated. It is the most common functional testing type and performed as
automated testing. \\
 \\
Generally, developers build different modules of the system/software
simultaneously and don’t focus on others. They perform extensive
[[dido:public:ra:xapend:xapend.a_glossary:b:blackboxtesting|black box]] and
[[dido:public:ra:xapend:xapend.a_glossary:w:whiteboxtesting|white box]]
functional
[[dido:public:ra:xapend:xapend.a_glossary:v:vendorlockin|verification]],
commonly known as [[dido:public:ra:xapend:xapend.a_glossary:u:unittesting|unit
tests]], on the individual modules. Integration tests cause data and
operational commands to flow between modules which means that they have to act
as parts of a whole system rather than individual components. This typically
uncovers issues with UI operations, data formats, operation timing,
[[dido:public:ra:xapend:xapend.a_glossary:a:api | API]] calls, and database
access and user interface operation.\\
 \\
Source: [[https://www.simform.com/functional-testing-types/]]
</WRAP> |
| [[dido:public:ra:xapend:xapend.a_glossary:i:interfacetesting]]   |
<WRAP>**Interface Testing** is defined as a software testing type which
verifies whether the communication between two different software systems is
done correctly. \\
 \\
A connection that integrates two components is called interface. This
interface in a computer world could be anything like
```

```
[[dido:public:ra:xapend:xapend.a_glossary:a:api]], web services, etc. Testing
of these connecting services or interface is referred to as Interface Testing.
\\
 \\
An interface is actually software that consists of sets of commands, messages,
and other attributes that enable communication between a device and a user. \\
 \\
Source: [[https://www.guru99.com/interface-testing.html]]
</WRAP> |
| [[dido:public:ra:xapend:xapend.a_glossary:r:regressiontesting]]  |
<WRAP>**Regression Testing** is re-running tests for
[[[[dido:public:ra:xapend:xapend.a_glossary:f:funcreq]]]] and
[[dido:public:ra:xapend:xapend.a_glossary:n:nonfuncreq]] to ensure that
previously developed and tested software still performs after a change. If
not, that would be called a **regression** as far as functionality. Changes
that may require regression testing include bug fixes, software enhancements,
configuration changes, and even substitution of electronic components. As
regression test suites tend to grow with each found defect, test automation is
frequently involved. Sometimes a change impact analysis is performed to
determine an appropriate subset of tests (non-regression analysis).  \\
 \\
Source: [[https://en.wikipedia.org/wiki/Regression_testing ]]
</WRAP> |
| [[dido:public:ra:xapend:xapend.a_glossary:s:sanitytesting]]      |
<WRAP>**Sanity Testing** is a subset of
[[dido:public:ra:xapend:xapend.a_glossary:r:regressiontesting|regression
testing]]. Sanity testing is performed to ensure that the code changes that
are made are working as properly. Sanity testing is a stoppage to check
whether testing for the build can proceed or not. The focus of the team during
sanity testing process is to validate the functionality of the application and
not detailed testing. Sanity testing is generally performed on build where the
production deployment is required immediately like a critical bug fix. \\
 \\
Source: [[https://www.geeksforgeeks.org/sanity-testing-software-testing/]]

</WRAP> |
| [[dido:public:ra:xapend:xapend.a_glossary:s:smoketesting]]       |
<WRAP>**Smoke Testing** is performed on the ‘new’ build given by developers to
QA team to verify if the basic functionalities are working or not. It is one
of the important functional testing types. This should be the first test to be
done on any new build. In smoke testing, the test cases chosen cover the most
important functionality or component of the system. The
[[dido:public:ra:xapend:xapend.a_glossary:o:objective|objective]] is not to
perform exhaustive testing, but to verify that the critical functionality of
the system is working fine. \\
 \\
Source: [[https://www.simform.com/functional-testing-types/]]
</WRAP> |
```

```
| [[dido:public:ra:xapend:xapend.a_glossary:u:unittesting]]        |<WRAP>
**Unit Testing** ensures that each part of the code developed in a component
delivers the desired output. In unit testing, developers only look at the
interface and the specification for a component. It provides documentation of
code development as each unit of the code is thoroughly tested standalone
before progressing to another unit. \\
 \\
Unit tests support functional tests by exercising the code that is most likely
to break. \\
 \\
Source: [[https://www.simform.com/functional-testing-types/]]
</WRAP>|
| [[dido:public:ra:xapend:xapend.a_glossary:w:whiteboxtesting]]    |
<WRAP>**White Box Testing** techniques analyze the internal structures the
used [[dido:public:ra:xapend:xapend.a_glossary:d:datastructure|data
structures]], internal design, code structure and the working of the software
rather than just the functionality as in
[[dido:public:ra:xapend:xapend.a_glossary:b:blackboxtesting|black box
testing]]. It is also called glass box testing or clear box testing or
structural testing. \\
 \\
Source:
[[https://www.geeksforgeeks.org/software-engineering-white-box-testing/]]
</WRAP> |

/**=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
=-=-=-=-=-=-=-=-=-=-
/* To add a discussion page to this page, comment out the line that says
  ~~DISCUSSION:off~~
*/
~~DISCUSSION:on|Outstanding Issues~~
~~DISCUSSION:off~~
```