Types of Testing



Prepared by

**Fasttrack**

Solution Architect

Contributors

**List of Contributors**

Revision and Signoff Sheet

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
|  |  | 1.0 | Initial draft for review/discussion |
|  |  |  |  |
|  |  |  |  |

Reviewers

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version Approved | Position | Date |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

Table of Contents

[1 Introduction 5](#_Toc128737888)

[1.1 Unit Testing 5](#_Toc128737889)

[1.2 Integration Testing 6](#_Toc128737890)

[1.3 System Testing 6](#_Toc128737891)

[1.4 User Acceptance Testing 6](#_Toc128737892)

[1.5 Regression Testing 7](#_Toc128737893)

[1.6 Performance Testing 7](#_Toc128737894)

[1.7 Security Testing 8](#_Toc128737895)

1. Introduction

There are many types of testing that can be performed throughout the software development lifecycle, depending on the goals of the testing activity and the nature of the software being developed.

This document will outline the many different types of testing.

* 1. Unit Testing

While not authored and executed by testers, unit and integration testing play an important role in ensuring the quality of an application.

The benefit of unit and integration testing is that these techniques can detect Defects much earlier than tests authored and executed by testing teams. Additionally, since these tests are automated and generally execute quickly, they are terrific for regression testing purposes and can be leveraged as part of automated build processes (see Smoke Testing).

In many cases, organizations do not distinguish between unit and integration testing, because they are both coded with the same tool (Visual Studio) and are both executed in in the same ways (Visual Studio or as part of an automated build process).

Unit test authoring and execution would take place in any sprint in which there is custom-coding activity. It is considered best practice to write these tests at the same time as the code under test is being created or modified. Some organizations follow an approach of writing unit tests prior to writing the application code, which is called Test-Driven Development (or TDD).

A unit test is an automated test (a small piece of code) that is testing the smallest testable piece of a software product in isolation. This smallest testable piece may be a single method or class. The practice of unit testing can be described as “writing code to test your code”.

Unit tests not only execute the code under test (Act) but also automatically check (Assert) the outcome of the test automatically. Human interaction is not needed to determine if the code under test behaved as expected. A key benefit of unit tests are that they can easily and repeatedly be executed to quickly find out if the code under test still behaves as it should. This makes unit tests a critical part of implementing regression testing, which is the practice of ensuring that recent changes to the code under test do not introduce new and unexpected defects.

The isolation aspect unit testing refers to eliminating any dependencies that the unit may have outside of itself. This isolation can be achieved through techniques such as mocking and stubbing, which are facilitated in Visual Studio 2010 through a tool called Pex and Moles and in Visual Studio 2012 and 2013 through a tool called Fakes.

The responsibility for conducting unit testing lies with the development team. Code coverage data can be collected to give an indication of how much of the code of the application under test has been covered by the tests executed.

* 1. Integration Testing

Integration testing is the verification that distinct components (i.e. groups of code units) of the application work in accordance with the user requirements. For integration testing, it is acceptable to directly utilize dependencies such as databases, other software components, and even external systems. For this reason, an integration test can be viewed as being a “relaxed” unit test.

The authoring and execution of integration tests are done in the same way as unit tests, so the real distinction is a relaxation of unit testing’s rules around usage of dependencies. Just like unit tests, the responsibility for conducting integration testing lies with the development team. Code coverage data can also be collected.

* 1. System Testing

System Integration Testing (SIT) is the last set of internal test activities covering the end-to-end functional scenarios tied with integrations with other systems.

SIT requires a careful planning such that all participating systems are available and test data is

* 1. User Acceptance Testing

User Acceptance Testing (UAT) is performed during the demonstration at the end of a non-release sprint. During the demonstration the MS Team presents the completed user stories to the Client team and demonstrates that the acceptance criteria for each user story has been met. After acceptance of the user story, the user story is considered to be completed.

* 1. Regression Testing

There are two types of regression testing that can be associated with a project.

* If the project is a new project, there is nothing to regression test initially. However, if a problem is discovered during testing and new code is delivered, all functionality that may be affected by the new code must be regression tested to make sure the fix didn't break something else. Any regression testing required because of problems discovered during the testing will be documented as part of the test summary and will not require an update to the test strategy or test plan.
* If the release is an update to an existing product, regression testing must be done to the current application to validate that no functionality has been lost due to the release. The extent of the regression testing must be identified as part of this section. Not all aspects of the code may need to be regression tested, depending on the release. If a problem is discovered during testing and new code is delivered, all functionality that may be affected by the new code must be regression tested to make sure the fix didn't break something else. Any regression testing required because of problems discovered during the testing will be documented as part of the test summary and will not require an update to the test strategy or test plan.
  1. Performance Testing

A vital aspect of an application’s success is not only doing what is expected of it in terms of functionality, but also in terms of response time and system throughput - commonly called performance.

Performance testing is used here as an umbrella term which encompasses a number of different specific types of tests. Within Visual Studio, performance testing is done using a load test. A Visual Studio load test enables the configuration of key parameters necessary for the execution of a number of different styles of performance tests.

Visual Studio load tests can contain any of the automated test types of Visual Studio, but the typical uses are with web tests and unit (or integration) tests. The results of load test execution can be saved to a database and extensive out-of-the-box and custom reporting is available to provide evidence of test execution and analysis of the results of the test.

Performance testing is usually conducted toward the end of the development lifecycle, when the application environment and business processes are considered stable. There is a growing movement to early lifecycle performance testing, which focuses on component level performance.

The MS Team conducts a Microsoft Dynamics Performance Review which involves a certain degree of performance testing to arrive at recommendations. Their recommendations are used to evaluate and drive changes to the application code itself, but are not used to drive infrastructure or topology changes, such as adding an additional web server or clustering the data tier. The reason this is not done is that usually the infrastructure for a project has been contractually defined prior to application development beginning.

For the best results from performance testing, the tests should be executed in an environment as close to production as possible. If this can’t be achieved by the client and MS Teams, the application can be brought into the lab of Microsoft’s ITSM practice where a production-like environment can be constructed and intensive performance tests can be conducted over a period of weeks. Without a production-like environment and the ability to alter infrastructure and its configuration, the ability to confirm the application’s performance against requirements and to drive improvements outside of the application code and its configuration will be limited.

* 1. Security Testing

The purpose of security within the context of CRM is to allow users to access data and perform specific actions in a controlled way. This is achieved through security roles working with permissions and access levels.

Microsoft CRM has a pre-defined set of privileges, which cannot be changed or added. Privileges correspond to actions that can be taken by the user upon entities or records.

Access levels in CRM define the entities and records upon which a user can perform actions. The level assigned to a user includes all levels below their assigned level of access.

Security roles in CRM bring together privileges and access levels for a specific job function that may be performed by multiple users. The privileges and access levels associated with pre-defined roles can be modified and new roles can be created.

During development sprints, the MS Test Team can author and execute manual tests using MTM (test results will be written to TFS and are available for reporting) or automated tests (which can be associated to test cases and executed by MTM) to verify that the security-related customizations are working as specified. During a release sprint, the MS Test Team can perform catch up work in authoring security tests and the full set of security tests should be executed.

The test data being used to perform functional testing should also include users assigned to security roles which have been added or customized from their out-of-the-box state.

Similar to functional tests, the effort to automate a security test should be weighed against the value it will deliver. To the extent that these tests can be automated with CUI tests, they should be and should be leveraged as part of automated build processes.