# Test plan writing guideline

# Preparation

### Identify the area which the test plan is about

Software product / High level area / Actual area

For example:

* AutoStore / Installation
* AutoStore / Process components / Barcode

### Extent

Planned execution time should be topmost 1 day. If a given test plan’s estimated execution time is more than 1 day, it would be separated to parts.

### File format

This guide is mainly for Excel files, but could be used for other tools as well.

Related test plans could be written to different tabs of the same Excel file.

# Format of the test plan

### Header

Record the environment of the actual execution. In case of AutoStore, there’s a template test plan header, which has to be used across all test plans.

* Product name and version number
* Tested component’s version number
* 3rd party engine version number (if available)
* Environment: OS and all the relevant tools (MFP, backend, database, web browser etc.)
* Execution date
* QA Engineer’s name
* Estimated and actual execution time
* Overall result, like number of PASSED test cases or a percentage of PASSED/FAILED/BLOCKED/NT results.

### 3rd party engines

Many AutoStore components are using built-in 3rd party engines, like Barcode, OPBarcode, Bates Stamp etc. In these cases we need to understand the difference:

* which parts are developed by the AutoStore team,
* how AutoStore uses the engine,
* which mistakes were made by AutoStore or the 3rd party engine.

These questions couldn’t be always decided at once. For the test plan writing phase:

* it’s important to ask for the 3rd party engine’s version number (if available)
* the AutoStore component’s own version number
* usually we need to test the AutoStore part, instead of the 3rd party engine.

### Parts of the test plan

* Tested higher-level areas, like chapters
* Tested sub-level areas like paragraphs
* Test cases which include Input procedure, Expected outcome, Evaluation, Notes
* Optional: priority (high/low), if a given part is more or less important than the others. Priority could be added to high level areas, sub-level areas or even test cases.

Example: Installation test plan

* High-level areas: Pre-requisites, Clean Installation, Uninstallation, Upgrade scenarios
* Sub-level areas for “Clean Installation”: Install with the defaults, Change directory, Change components
* Test cases for “Install with the defaults”: Start installer, Accept license agreement, Click Next, Click Back etc.

Example: Barcode component test plan

* High-level areas: Include to a workflow, General, Field Values, RRTs
* Sub-level areas for “Include to a workflow”: Create new Task, Setup the Task, Run the Task
* Test cases for “Create new Task”: Click New button, Type a name etc.

### Coverage

The test plan (when it is finished) should cover every related parts of the given area:

* pre-requisites (license, workflow, create/collect sample files)
* user interface (language, fonts, colors, hotkeys, tab order)
* functionality (every available option is in effect)
* important/likely combinations of the options
* environment-related parts: where not every test case is executable (different parts for different backend versions)

Example: Installation

* Pre-requisites: .Net 4.5 Framework, QA Engineer should be able to generate licenses
* User interface: installer’s dialogs, language, messages
* Functionality: install with defaults, Back buttons, change directory, change components, registry entries, licensing, basic functional test after the installation, uninstallation
* Important combinations: missing (1) license for an installed (2) component

Example: Barcode

* Pre-requisites: installed build, added license, available sample files
* User interface: configuration dialog design, it is clear, understandable, logical, hotkeys, tab order
* Functionality coverage: Barcode component in the ToolBox, integrate it to a workflow, General tab settings, Field Values settings, RRTs
* Important/likely combinations: setup zones + field values + RRTs

# Test cases

A given test case should include these parts: Input procedure, Expected outcome, Actual result, Notes.

### Input procedure

Input procedure is a short list of activities to be taken to get an interpretable result. It’s preferably one short sentence.

The content of the test cases are related to each other. The later test cases assume the execution of the previous test cases.

The language of an Input procedure has to be

* Clear
* Concise
* Understandable for less-practiced testers as well
* Contain all the information what needed to execute the very same scenario

### Expected outcome

What should be experienced after the input procedure was executed. Expectations:

* It has to be very clear
* It’s a fact about what is expected to happen, not a call or presumption
  + Good example: “The dialog disappeared”
  + Bad example (call): “Check that the dialog disappeared”
  + Bad example (presumption): “The dialog should disappear”
* It could be a list of facts, not necessarily only one thing. Good example:
  + The dialog disappeared
  + The directory was deleted
  + E-mail notification arrived to the user

### Evaluation

Leave blank, tester will fill it with the actual result, which could be

* PASSED , if the expected outcome happened
* FAILED, if the actual outcome was different than the expected (usually a result of a bug)
* BLOCKED, if the test case couldn’t be executed (for example the backend is not available). The executor must indicate the blocking issue in the Notes column.
* NOT TESTED, if the test case was not executed in any other reason. The reason has to be identified again in the Notes column.

### Notes

Leave blank, tester will fill it if needed. It could be

* a bug number, if the test case failed and the tester reported a bug about it,
* a bug number, if the test case failed and a bug was already reported about it,
* change/improvement suggestions for the test case (for example if the product changed since the test plan was written),
* any other questions or problems what the tester faced during execution.

### Types of test cases

* Positive: using the software in a similar manner as the customer/administrator/end user, with valid values and likely scenarios

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Run the installer with pre-installed .Net 4.5 framework | Installer started |
| Accept license agreement Click Next | The Custom setup dialog appeared |
| Install the software to the default Program Files directory | Installation was successful |

* Negative: causing errors or take unusual activity purposefully, with invalid data or unlikely scenarios. The expected outcome is usually a precise error message. Examples for negative test cases:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Run the installer without pre-installed .Net 4.5 framework | A proper error message appears. |
| Cancel installation before it was finished | A proper error message appears. |
| Add invalid password at backend configuration | A proper error message appears. |
| Type letters to a field which should accept only numbers | A proper error message appears. |
| Type an invalid string to freely-editable field (like Type on Barcode) | A proper error message appears. |

* Boundary values: using the extremes of the input domain. Inputs could be valid or invalid values as well. It’s most usual if the accepted input values are numbers only.  
  Examples for “CPU usage” value, which is valid between 0-100%:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Type 0 and click OK | Value was saved |
| Type 100 and click OK | Value was saved |
| Type 101 and click OK | A proper error message appears |

* Use case: a set of related, positive test cases. Describes a scenario, which is similar how the customers will use the software. For example, a short use case about the installation:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Start installer | InstallShield Wizard appears |
| Accept license agreement  Click Next | Custom Setup dialog appears |
| Leave the default components selected  Click Next | Installation started  When it finished, installation successful message appears |
| Click Finish | Installation wizard disappeared |

* Functional tests: investigating *what* the system does. It is based on requirements specification or previous experience.

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Install AutoStore | Installation successful |

* Non-functional tests: investigating *how* the system works

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Install AutoStore | Installation finished in 2 minutes |

### Parameterized input procedure

It’s possible that some configuration steps are not specified in the test plan, those are only specified by the QA lead, before the actual execution started. These appear as *parameters* of the test cases. Examples:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Install the build to the specified Windows Operating System | Installation successful |
| Configure the specified backend server as a Route component | Configuration successful |
| Use the specified user credentials to configure the backend server. | Configuration successful |

### Environment-related expected outcomes

Sometimes the expected outcome depends on the actual environment where the test is executed. For example AutoStore’s program directory depends on whether the actual Windows is a 32 or 64bit version:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Run AutoStore installer Go to the Custom Setup dialog | The offered program directory is  “C:\Program Files\Notable Solutions\AutoStore 7\” on 32bit Windows “C:\Program Files (x86)\Notable Solutions\AutoStore 7\” on 64bit Windows |

There could be a correlation between the parameterized Input procedure and the environment-dependent Expected outcomes. But the correlation is not necessary, it needs to be taken into account in every test cases one-by-one. For example:

* The expected outcome depends on the specified parameter:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Install AutoStore to the specified version of Windows OS | Installation was successful  AutoStore is installed to the…  “C:\Program Files\Notable Solutions\AutoStore 7\” directory in case of 32 bit Windows  “C:\Program Files (x86)\Notable Solutions\AutoStore 7\” directory in case of 64bit Windows |

* There is a specified parameter, but the expected outcome does not depend on it:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Install AutoStore to the specified version of Windows OS | Installation was successful  AutoStore appeared in the Start menu  AutoStore entries were saved to the registry |

* There’s no specified parameter, but the expected outcome depends on the QA engineer’s actual environment:

|  |  |
| --- | --- |
| **Input procedure** | **Expected outcome** |
| Install AutoStore | Installation was successful  AutoStore is installed to the…  “C:\Program Files\Notable Solutions\AutoStore 7\” directory in case of 32 bit Windows  “C:\Program Files (x86)\Notable Solutions\AutoStore 7\” directory in case of 64bit Windows |