# Analytics Products Testing Standards

This document outlines Analytics Products testing standards. Our goal in testing our work product is to ensure we deliver analytic algorithms that

1. are correct to the limits of machine precision
2. perform well
3. scale well
4. have self-service user interfaces.

We also want our testing practices to catch design and coding errors at the earliest possible opportunity, in the belief that doing so minimizes the waste that errors create.

These standards concern three types of test:

* A **unit test** exercises a single function.
* A **functional test** exercises an entire tool by embedding the tool in an Alteryx dataflow, executing the dataflow through the Alteryx server, and evaluating the tools’ behavior output.
* A **user-interface test** exercises an entire tool by embedding the tool in an Alteryx dataflow, executing the dataflow through the Designer user interface, and evaluating the tool’s presentation layer, behavior, and output.

We treat other types of tests as subtypes or aspects of the above classes. For example, a **regression test** is a functional or user-interface test that ensures a tool does not exhibit a previously reported functional or design bug.

Below we enumerate general testing standards, and standards for each type of test.

## General Testing Standards

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Description** | **Purpose** | **Enforcement**  **Steps** | **Enforcement**  **Mode** | **Ratification**  **Date** |
| T1 | The tests written for a given tool constitute its acceptance criteria (executable specification). |  |  |  |  |
|  |  |  |  |  |  |

## Unit-Testing Standards

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Description** | **Purpose** | **Enforcement**  **Steps** | **Enforcement**  **Mode** | **Ratification**  **Date** |
| UT 1 | Code unit tests for each function’s happy, alternate, and sad paths. |  |  |  |  |
| UT 2 | Unit test only for correctness. |  |  |  |  |
| UT 3 | Unit tests not requiring database input or output should execute in five seconds worst case and 0.01 seconds or less on average. |  |  |  |  |
| UT 4 | Unit tests requiring database input or output should execute in 10 seconds worst case and 0.1 seconds or less on average. |  |  |  |  |
| UT 5 | Check in all of a function’s unit tests before merging the function into any build. |  |  |  |  |
| UT 6 | Manually execute a function’s unit tests before checking in the function. Only check in a function that passes all unit tests. |  |  |  |  |
| UT 7 | Automate execution of unit tests upon code check in. |  |  |  |  |
| UT 8 | Automate execution of unit tests during builds. |  |  |  |  |
| UT 9 | Use [tool] for R unit tests. |  |  |  |  |
| UT 10 | Use [tool] for C++ unit tests. |  |  |  |  |
| UT 11 | Use [tool] for Python unit tests. |  |  |  |  |
| UT 12 | Use [tool] for Scala unit tests. |  |  |  |  |