Test plan overview

This is the test plan for the Privacy protection for information control application requested by SINTEF ICT. This test plan will be based on IEEE829-1998, the IEEE standard for software test documentation, with some adaptions to fit this project better. The purpose of testing is to find bugs and errors and correct them, and to make sure the program is working as expected. The purpose of this test plan is to make sure the tests will be executed as planned, and that they are well documented.

0.1 Test Methods

There are two main types of software testing: black-box testing and white-box testing.

Black-box testing

This is a method that test the functionality of an application. For this type of testing, knowledge about the application's code and structure is not required. The test cases are based on external descriptions of the software, e.g. specifications, functional requirements or designs. Black-box tests are usually functional, but they can also be non-functional. This type of testing can be applied to all levels of software testing.

White-box testing

This method is used for testing internal structures of an application. For white-box testing it is required to have both knowledge about the code and structure of the application, as well as knowledge about programming to design the test cases. This type of testing is normally done at unit level where it can test paths within a unit or paths between units, but it can also be used at integration and system levels of testing. This method can uncover many errors and problems, but it is not a good test method for finding out whether the program is fulfilling the requirements or not.

0.2 Testing approach

The main focus will be on white-box testing. This is a program that is going to be used for research, which means that black-box testing will not be very useful as the client want to work on and test algorithms themselves. The main task is to deliver a good framework with the necessary tools, and a working, learning algorithm so that further testing can be done with ease. Since one of the system requirements is high modularity, it will be a goal to have the tests be as little dependent on other modules as possible. There will be no training needs, as the testers are also involved in the programming.

What will be tested:

- **Unit testing:** This will be used for testing the functionality of the modules, so that it can be ensured that every module is working as intended.
- **Interface testing:** As the algorithm is based on case-based reasoning (CBR), it will be important to test that it is learning from new data.

What will <u>not</u> be tested:

- **Usability testing:** This program is intended for further research by the client, and not for use by customers. Since this program is not supposed to be ready for end-users, there is no need to perform end-user tests to see how the users interact with the program, and whether the program is accepted by users or not.
- **Graphical user interface (GUI) testing:** For the same reasons there will not be any tests on the quality of the GUI. The GUI that will be included is there to make testing easier for the client, not to provide the best possible interaction with end- users.
- **Run time:** There will be no designated tests for checking and optimizing the run time of the algorithm. This is because the main classification algorithm can be easily changed with another algorithm. Run time will only be looked into if the program is very slow even for small data sets.

0.3 Test case overview

Under is the test cases and their identifiers. The identifiers are named UNIT-XX, where XX is the number of the test case.

Unit tests:

- UNIT-01: Command line interface (CLI) functionality
- UNIT-02: P3P parser
- UNIT-03: Local database
- UNIT-04: Graphical user interface (GUI) functionality
- UNIT-05: Algorithm classification
- UNIT-06: Algorithm learning
- UNIT-07: Packet passing through network to community database

Under is the test case template for the unit tests

Item	Description
Name	The name of the test
Test identifier	The identifier of the test
Person responsible	The person responsible for making sure the test is executed correctly and on time.
Feature(s) to be tested	What kind of functionality that is being tested.
Pre-conditions	What code and environment that has to be in place before the test can be executed.
Execution steps	Stepwise explanation of how to perform the test.
Expected result	The expected output/result for the test to be successful.

Under is the template to be used when executing a test.

Item	Description
Name	The name of the test
Test identifier	The identifier of the test
Person responsible	The person that executed the test
Date of execution	The date of when the test was executed
Execution steps	Stepwise explanation of how to perform the test.
Steps executed	The steps executed by the tester.
Expected result	The expected output/result for the test to be successful.
Step results	The results of the performed steps. SUCCESS / NO SUCCESS for each step.
Test conclusion	SUCCESS / NO SUCCESS for the entire test. Only successful if every step is successful.

0.4 Test case overview

See appendix for the test cases.

0.5 Test pass / fail criteria

A test is passed if the given execution steps and input produce the expected results. If they do not, the test is failed.

0.6 Test schedule

Under is the table with the scheduled execution dates of the unit tests.

Test identifier	Execution date
UNIT-01	Saturday 22.10
UNIT-02	Saturday 22.10
UNIT-03	To be determined
UNIT-04	To be determined
UNIT-05	To be determined
UNIT-06	To be determined
UNIT-07	To be determined

0.7 Risks and contingencies

For some of the tests it is not possible to test every possible input / output because there is too many different combinations of input. So there is a chance a test will pass all the combinations used in the test cases, but still fail at some later point for some other combination. This will be a problem for the tests UNIT-02 P3P parsing, and UNIT-05 Algorithm classification.

The P3P policies have a huge variety in which elements they contain, and certain elements have a N-to-1 relation, so it will be impossible to test if everything is parsed correctly for every possible P3P policy. The best way to prevent this is to handpick a necessary amount policies that are as different as possible, so that as many of the extremes as possible will be covered.

The algorithm classification tests have the same issue. There is simply too many combinations of learning base and input to cover everything. So these tests will have to be performed in such way that as many of the extremes as possible is covered.