Test Plan

PMoC
14.4.2003
Version 1.7

T-76.115 Test Plan
PMoC

Version	Date	Author	Description
1.7	13.04.2003	Jakobsson	Corrected some flaws in the document , no new chapters
1.6	20.03.2003	Jakobsson	Added ch. 3.4
1.5	07.02.2003	Jakobsson	Added ch. 3.3
1.4	04.02.2003	Jakobsson	Added testing for second test phase in ch. 3.2. Added fault injection & ad-hoc testing in ch. 5.
1.3	24.01.2003	Jakobsson	Added features to be tested in ch. 3.2 & added regression test suite description to chapter 15.
1.2	2.12.2002	Jakobsson	Added features to be tested in ch. 3.1
1.1	1.12.2002	Jakobsson	Added chapters 6 & 7
1.0	18.11.2002	Jakobsson	First release
0.1	10.11.2002	Jakobsson	Draft

Table of Contents

1. Introduction

1.1 Purpose and scope of the test plan

The purpose of this test plan is to describe the testing practices used by PMoC and to identify the items to be tested during this project. Writing the test plan is a method for planning and managing testing and the plan will function as a guide for the group during testing activities. The plan will reflect the decisions made by the group regarding the testing in the software project. The plan will contain the features to be tested, how they will be tested and who will do the testing.

1.2 The product and environment

The product to be tested is the ProConf system. The system and the environment are described in the project plan.

1.3 Summary of test items

The test items can be divided into two categories:

- The gml package will be tested with unit and integration tests
- The graphics package will mostly be tested with unit and system tests.

1.4 Referenced documents

- V-model testing method
- Technical specification
- Project Plan
- Glossary

1.5 Terminology

The Glossary document defines the terms and definitions used in this document.

1.6 Document overview

The first part of the document (chapter 2 through 4) contains a detailed description of items and features that will be tested. The plan then addresses the testing approach and different criteria for testing (chapter 5 through 8). The end chapters deal with managing the testing activities in the project, including responsibilities, schedules and risk management.

2. Test items

All modules of the project are test items. The modules and the classes are described in detail in the technical specification. The Batik component will not be tested, as it is developed by Apache.

3. Features to be tested

The features to be tested include the user requirements described in the requirements document and also several non-functional requirements. During the project different features will be tested.

3.1 First implementation phase

The first implementation phase will produce a prototype for the ProConf system. During this phase we will mostly do unit testing and integration testing. The gml package should be able to parse a gml data file and build an internal representation of the data. The graphics package should be able to render an svg file and allow modification of elements, such as changing the coordinates of an element. The exact features to be tested will be described in the test report document. The priority of the test items are 1 for high, 2 for medium and 3 for low.

Table 1: Features to be tested in phase I1 with unit tests

Module to be tested	Feature to be tested	Priority
fi.vtt.proconf.graphics.*	Drawing svg elements	1
	Moving svg elements	2
	loading an svg file	1
	saving SVG graphics to file	2
fi.vtt.proconf.gml.*	parsing of gml	1
	loading gml data into an internal representation.	2
	performance (speed)	3
fi.vtt.proconf.browser.*	showing gml data in the tree view	1

Table 2: Features to be tested in phase I1 with integration tests

Packages to be tested	Feature to be tested	Priority
fi.vtt.proconf.graphics.*	Combining the graphics tools with the canvas. Forms a primitive graphics editor.	1
fi.vtt.proconf.gml.* fi.vtt.proconf.browser.*	Integrating the gml classes to allow showing contents of a gml project in the browser	1

3.2 Second implementation phase

The testing in the second phase is divided into two main test sessions. During week 4 the symbol editor will be tested. At this point the symbol editor will consist of almost all the features that will be in the final product. Testing is conducted on all levels, including system tests for the symbol editor. During the second testing period, which takes place during the last week of phase I2, basic functionality for the neteditor will be tested. At this point the gml and graphics parts of the system should be communicating with each other and with the io-package.

The tests in the first part are mostly system tests for the symbol editor. The name of the use case in the user requirements document that corresponds to the test case is included in the description.

Table 1: Features to be tested during the first test session in phase I2

Items to be tested	Feature to be tested	Priority
fi.vtt.proconf.graphics.*	**System tests:**	
	STC_A1: New symbol	1
	STC_A2: Open symbol	1
	STC_A3: Import graphic	1
	STC_A4: Save symbol	1
	STC_A5: Add graphical object to symbol	1
	STC_A6: Change graphical attribute	2
	STC_A7: Move graphical object	1
	STC_A8: Delete graphical object	1
	STC_A17: Scroll symbol	2
	STC_A18: Zoom symbol	2
all subclasses of fi.vtt.proconf.graphics.Tool	Adhoc testing: The graphical drawing tools	1
fi.vtt.proconf.gml.* fi.vtt.proconf.browser.*	Regression tests with old unit tests	1

Table 2: Features to be tested during the second test session in phase I2

Items to be tested	Feature to be tested	Priority
fi.vtt.proconf.graphics.*	**System tests:**	
	STC_A16: Add terminal	1
	STC_A25: Attach symbol to type	1
	STC_A26: Make free symbol to a terminal symbol	1
	STC_B2: Scroll net	1
	STC_B3: Zoom net	1
	STC_C1: New component	1
	STC_B1: Open net	1
	STC_C2: Save net	1
	STC_C3: Add graphical object to net	1
	STC_C4: Move graphical object	1
	STC_C5: Delete graphical object	1
	STC_C6: Add component to net	1
	STC_C7: Move component symbol	1
	STC_C8: Delete component	1
	STC_C9: Change component property or attribute value	1
	Integration tests:	
	Loading data through the io-package	2
fi.vtt.proconf.gml.* fi.vtt.proconf.browser.*	**System tests:**	
	STC_A9: Add property	1
	STC_A10: Change property name	2
	STC_A11: Change property type	2

	STC_A12: Delete property type STC_A13: Change property attribute STC_A14: Edit default value of property STC_A15: Add subtype to type STC_A19: Save component type **Integration tests**: Loading data through the io-package **Unit tests:** Regression tests	1 2 1 1 1 2 1

3.3 Third implementation phase

During the third implementation phase, the system test run in phase I2 will be repeated and the system should pass these tests in the browser too. In addition to that the new system tests are included in table 3 below.

Table 3: New features to be tested during phase I3

Items to be tested	Feature to be tested	Priority
fi.vtt.proconf.*	**System tests:** STC_A21: Import component type STC_A22: Create component type STC_A23: Add constraint STC_A24: Remove constraint STC_A27: New connection symbol STC_C10: Add connection STC_C11: Delete connection STC_C12: Change connection start/end terminal STC_C17: Save component STC_D1: Create category STC_D2: Change category attribute STC_D3: Delete category **Integration tests:** Loading and saving data through the io-package **Unit tests:** Regression tests with unit tests	 1 1 1 1 1 1 1 1 1 1 1 1 1 2

3.3.1 Peer Testing

The peer testing will be conducted during week 10 by Openlogbook and we will test their system on week 9. During peer testing the installation and use of our system will be tested along with more user oriented testing. We do not have the resources to educate the peer group in the proconf system and the gallery data model, so specific functionalities and test cases will not be tested by the peer group. The basic use cases in the user requirements document will be given to the peer group and they will get instructions for doing the basis tasks such as drawing components and constructing nets. The gml data structures will not be tested by the peer group, and the browser will be tested only minimally. If the peer testing is successful we might get valuable insight in how usable our program is, even though our primary target group has much more thoroughly knowledge of the system. There is a risk that some feedback from the peer group will be based on misunderstanding or lack of knowledge, but we'll have to live with that. We try to minimize the risk by not going into details. We will deliver our system to the peer group through our on-line phorum.

3.4 Delivery phase

For the delivery phase testing is concentrated to regression tests, including all previously made tests that are still applicable. As open bugs are fixed the program will be altered in many modules and it is important that we run the tests to be certain that the program is still intact. A summary of the tests are presented in table 4.

Table 3: System features to be regression tested during delivery phase.

Items to be tested	Feature to be tested	Priority
fi.vtt.proconf.*	**System tests:**	
	Symbol editing:	
	STC_A1-2, 4-8, 16-17, 21, 25-27	1
	STC_A3, 18, 20	3
	Net canvas viewing	
	STC_B1-2	1
	STC_B4	2
	STC_B3, 5, 6	3
	Net editing	
	STC_C1-11, 17	1
	STC_C12-13	2
	STC_C14-16	3
	Browser editing	
	STC_D1-3	1
	STC_D4,6	3
	Integration tests:	
	Loading and saving data through the io-package	1

	Testing drawing tools	1
	Unit tests:	
	GML datastructure unit tests	1

4. Features not to be tested

All features cannot be tested within a reasonable amount of time. Therefore we will most likely have to leave some features untested. The most important features will be tested first, and the features with the lowest priorities can be dropped from testing if no time is available.

5. Approach

The testing approach is described in detail in the V-model testing method document.

5.1 Fault injection

To improve the coverage of the testing, we run the test not only the way the are supposed to run, but also always try something that isn't supposed to work. Fault injection is used from unit test up to system tests. Some unit tests are explicitly written with destructive intentions, while system tests include fault injection without it being separately stated in the test case or test report. The test results reveal what kind of fault injection caused the test to fail.

5.2 Ad-hoc testing

Because the proconf system consists of graphical editors with many tools and function, ad-hoc testing is an efficient way of testing these tools. Writing test cases for all the different tools would have a better coverage, but would require a huge amount of time. Therefore ad-hoc-testing is used as a supplemental testing method.

6. Item pass/fail criteria

6.1 Fault levels

System and acceptance tests have different levels of faults. These are described in the table below.

Table 5: Fault categories

Fault category	Description
Critical	A vital part of the system is not functional.
Serious	Part of the system has a malfunction, causing errors with vital parts of the system in certain situations.
Normal	A non-vital feature of the system is not functioning

	properly.
Low	A bug that does not affect functionality.

When a bug is discovered in the system and it cannot be fixed during the current coding or testing session, it is entered into the Burana bug tracking system. The bug is given an internal priority, as shown in the table below. The table also states the required actions when a bug is encountered. Every bug has a responsible person.

Table 6: Internal priorities for bugs

Internal priority	**Explanation and required action**
Low	The bug is fixed if there is time and no higher priority bugs or features are left. Usually corresponds to fault level Low.
SemiLow	The responsible person attends to the problem, but the rest of the group can continue as normal. Usually corresponds to fault level Normal.
Before release	The same as the Low or Semilow, but the bug has to be fixed before a release.
High	The development of the part of the system that is not functioning properly is frozen, and all efforts of persons involved with the subsystem are directed to fixing this bug. Usually corresponds to fault level Serious.
Urgent	The bug must be fixed immediately, before developing any other parts of the system. Usually corresponds to fault level Critical

6.2 Unit and integration test criteria

Unit tests and integration tests pass only if the result is as expected. The tests fail if the result deviates from what is expected.

6.3 System and acceptance test criteria

A test is accepted even if it contains a fault of low category. Low category faults (for example small visual bugs) are corrected only if the customer sees them as a real nuisance and there is time to correct them.

7. Suspension criteria and resumption requirements

Testing can be suspended after completing test without any special criteria. Testing is later resumed with the rest of the tests. This allows testing to be done whenever there is something to test, and makes testing an activity that is constantly performed in parallel to programming.

While running a test, testing can be suspended if the system has a fault that prevents the test from being run or if the environment cannot be setup so that the test can be run. Testing is resumed only when the reason for suspension is found and the fault is corrected.

8. Test deliverables

Test case specification is integrated into the test report for clarity. The test case description and the results are better viewed together in the same document. The test cases used in this project are different depending on the tests. Unit tests consist of a short textual description. Integration tests include the preconditions, steps and expected result of the test case. The test cases for system tests matches their corresponding use cases in the requirements document, and will not have a separate test case. This way updates are easier and no duplicate information will be written.

The following documents will be produced:

- Test plan – updated in every phase
- Test report – test case descriptions and results
- Log file – test results of unit tests

9. Testing tasks

Testing tasks are described in detail in the V-model testing method document.

10. Environmental needs

The testing environment is the same as the development environment, described in the project plan.

11. Responsibilities

Responsibilities are described in detail in the V-model testing method document.

12. Staffing and training needs

Testing will require some training by every tester, but the information will be gathered individually during the project without any special training needs. No staffing needs are present, since the group can muster all the required resources.

13. Schedule

Unit tests are run while developing a unit. Unit test are also run in integration testing to find out if new features has uncovered bugs in finished modules. System testing is performed when the corresponding parts of the system is ready. The only real milestone for the testing is that development is frozen a week before deadlines, so that there is time for final system testing bug corrections.

The table below illustrates how the testing is scheduled over the phases.

Table 7: Planned effort - hours to be spent on testing

	Tomas Backas	Johan Forsbäck	Björn Forss	Kenneth Haglund	Anders Gebala	Markus Jakobsson	Jan Lönnberg	Total
PP						5		5
I1	4	0	5	2	10	8	2	31
I2	3	3	7	3	6	10	3	35
I3	3	3	8	3	8	12	3	40

DE	2	2	3	2	3	7	2	20
Total	12	8	23	8	27	42	10	130

14. Risks, contingencies and assumptions

The table below suggests that lack of test items is the greatest risk associated with testing. The relatively high probability is due to the fact that the domain is unknown and it might take a while to produce the first modules of the project.

The severities of the risks are graded on a scale from 1 to 5 where 5 is the most severe impact on the testing of the project.

Table 8: Risks at the beginning of phase T1

Risk	Probability	Severity	Reaction	Prevention
Lack of time	40 %	3	Drop low priority tests	Start early, allocate time better.
No test items	33 %	5	Divert resources to implementation.	Early start, small increments, small modules.
Testing is ineffective	20 %	4	Redesign tests and test again.	Improve the coverage of test cases.
Planning mistakes	15 %	2	Improvise and decide on new plans.	Plan carefully, do research.
Incompetence	10 %	1	Study and learn, allocate more time to testing	Compare difficulty and staffing needs with available resources in planning.

15. Test suites

The unit tests will be gathered into a test suite for regression testing and will be run automatically. The results of the test run will be written to a log file.