Software Engineering Group Projects – Test Procedure Standards

Author:	H.R. Nicholls
Config. Ref.:	SE.QA.06
Date:	17 Oct 2000
Version:	3.7
Status:	Release

Department of Computer Science,
University of Wales,
Aberystwyth,
Dyfed SY23 3DB,
U.K.

CONTENTS

1 INTRODUCTION

1.1 Purpose Of This Document

The purpose of this document is to provide an aid to the production of good quality test documentation by software engineering group projects.

1.2 Scope

This document specifies the standards for writing software test plans, specifications and reports. It describes the necessary layout and content of these documents. It does not prescribe any particular method of testing such as top-down or bottom-up, but it does indicate the main stages of testing which should be carried out.

This document should be read by all project members. It is assumed that the reader is already familiar with the QA Plan [1].

1.3 Objectives

To describe the format of, and information which must be supplied in:

- test plans
- test specifications
- test reports

Also to illustrate how testing strategy drives the production of these documents, and to provide a framework for their production.

2 RELEVANT QA DOCUMENTS

The Test Plan, Specification and Report must be produced in accordance with the quality standards specified in the QA Plan [1]. In particular, they must be produced as part of specified tasks in the Project Plan [2] and maintained within the configuration management system according to the appropriate procedures [4]. The basic layout and information content must conform to the general documentation standards [3].

3 GENERAL APPROACH TO TESTING

Testing is used to establish the presence of defects in a program and it is also used to estimate whether or not a program is operationally usable [6]. It is important to remember that testing can only demonstrate the presence of errors — it cannot prove their absence. Testing is thus used to 'detect but not correct' — the function of testing is the discovery of errors, and the correction of errors should be left until *all* of the tests specified have been conducted. If errors are discovered, then the specified problem reporting and change control procedures must be followed in order to correct them [4]. Then, *all* the tests which exercise a system containing the amended items must be repeated. This is known as *regression testing*.

Particular emphasis should be placed on testing boundary situations, both inside and outside boundaries. For example, if a program is to process a maximum of 100 items typed in by the user, then test the program's behaviour for 0, 1, 100 and 101 items input as well as for some number of items between 1 and 100.

Software developed on large projects is normally subjected to five levels of testing: unit, module, subsystem, system, and acceptance. Unit testing is where each component (e.g., a subprogram) is treated as a stand-alone item and is tested on its own. Module testing involves exercising a collection of related components (e.g., a package containing type definitions and subprograms), and the collection is tested in isolation from the rest of the system.

For the group projects, it is appropriate to merge unit and module testing into a single test stage conducted by the programmer, which will be known as module testing. Module tests can be carried out by the programmer of that component, and do not need formal test plans or specifications.

Subsystem testing involves exercising a collection of modules (e.g., a group of related packages used by a major subprogram) in order to discover errors in a localised part of the system. A typical project might conduct independent subsystem tests on the main data structure packages, on the application-specific packages and subprograms, and on the user interface packages.

System testing involves integrating all the subsystems together to form a complete system and then exercising that system.

It is desirable that subsystem and system testing is carried out by persons other than those involved in the design or programming of the modules being exercised[1]. However, for group projects it is recognised that this may not be possible since it may increase the workload significantly.

Acceptance testing involves exercising the entire system according to a set of procedures produced by the client, such that if all the tests are passed, the client agrees to accept the product. Acceptance testing is normally carried out by the client in the presence of all group members.

Sommerville provides some useful background information on testing in Chapters 22 and 23 of his textbook [6]. Pfleeger [7] also discusses both program testing and system testing in some detail.

[1]'Testing' in this context means the production of test plans and specifications, and the execution of tests.

4 TEST PLANS

4.1 Purpose

The purpose of a test plan is to prescribe the scope, approach, resources and schedule of testing activities. It must identify the items being tested, the tasks to be performed and the personnel responsible for each testing task. It does *not* describe the details of each test, since that information is described in the Test Specification.

4.2 Topics To Be Covered

A test plan shall cover the following topics:

Test items: each item to be tested must be identified by its name, configuration reference, and importantly, its version number.

Features to be tested: identify all features and combinations of features to be tested. Each feature should be cross-referenced to the appropriate section in the document from which tests are being derived, such as the requirements specification or the design specification.

For example, consider the test plan for a package which stores and manipulates dates. The test plan would identify the following features (there would be many more in a full test plan): the correct storage of the first two days of the earliest permitted year and the last two days of the latest permitted year, the correct calculation of the first and last days of each month (both in leap years and in non-leap years) and of various other legal combinations, and also the rejection of various illegal conditions such as attempting to store the 29th of February in a non-leap year.

Features relating to performance, such as response times, storage requirements, etc. would also be identified under this section.

Remember that the purpose of this section is to *identify* the features to be tested, and that the details of exactly *how* these features will be tested must be supplied in the Test Specification.

Approach: describe the overall approach to testing. Specify the major activities, techniques and tools which are used to conduct the testing. The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to do each one.

Specification of the approach to testing might include: stating that features will be tested first with valid data and then with invalid data; that all input data will be prepared in a file and input redirection used to effect the input; that a file of *expected* results will be prepared in the same format as that which should be produced by the program under test, thereby enabling automated comparisons of expected and actual results; that a program will be written to exercise the module being tested (i.e., a **test harness** will be developed), and so on.

Item pass/fail criteria: specify the criteria to be used to determine whether each test item has passed or failed testing.

One way of specifying pass/fail criteria is to write the tests as a series of actions followed by questions which must be answered in the affirmative for a test to be passed (e.g., "Type 'quit'; did the program terminate and the Unix prompt appear?" — if this question can be answered 'Yes', then the test was passed). The questions would be posed in a form such that an affirmative answer indicates that a particular requirement of the system (as described in the Requirements Specification) has been met.

Alternatively, a pass/fail criterion might be to state that if the (possibly automated) comparison of expected results with actual results yields an exact match, then the test was passed.

Test deliverables: identify the deliverable documents concerning testing, including the test plan, test specifications, and test reports.

Testing tasks: identify the set of tasks necessary to prepare for and to execute testing. Identify all intertask dependencies. Note that some test harness software may have to be written, and should be produced as the result of identifiable tasks.

Environmental needs for testing: specify the required properties of the testing environment, including the hardware, system software, and any other software needed to support the tests (e.g., "the testing must be carried out via an `xterm` window connected to `legion`, displayed on a Sun Sparc computer running the `twm` window manager").

Responsibilities: identify the persons responsible for carrying out each of the testing tasks.

Schedule: produce a schedule for conducting the testing tasks and include test milestones identified in the Project Plan. Define any additional milestones and estimate the time required for each test task and milestone.

4.3 Required Test Plans

Test Plans shall be written by the group for the following test stages:

- subsystem testing;
- system testing.

The Subsystem Test Plan must be derived from the Design Specification. The System Test Plan must be derived from the Requirements Specification and the Design Specification, since system testing is attempting both to verify and to validate the system.

5 TEST SPECIFICATIONS

5.1 Overview

The purpose of a test specification is to specify in detail each of the tests to be executed as part of a formal test process. The test specification must cross-reference to the appropriate section of the corresponding test plan for each feature being tested.

Each test specification shall have an introductory section followed by a collection of test procedures. The introductory section shall have the form as specified in QA Document SE.QA.03 [3], and must include a list of the documents from which the Test Specification is derived. Full bibliographic details of the documents can be provided in the reference list at the end of the document, but each document must be referenced in the introductory text.

Each individual test must be described in detail and uniquely identified; this description is known as a **test procedure**.

The content of a test procedure is specified in the next section.

5.2 Test Procedures

Each test procedure shall have the following structure:

Test identification: every test must have a unique identifier of the form:

SE–groupletter–spectype–id

where:

- groupletter is the unique upper case alphabetic character assigned to the group at the start of the project (e.g., 'J' for project group J);
- spectype is one of: SS (for a subsystem test specification), or ST (for a system test specification).
- id is a three digit number uniquely identifying each test within the specification, starting from 001 and using leading zeros when required.

A test identifier for the fourteenth test within the system test specification of project group J would thus be: SE–J–ST–014 .

Test items: identify the items to be exercised by the test, cross-referencing to appropriate documents (e.g, to the Subsystem Test Plan and to the Design Specification).

Test purpose: a brief (e.g., one sentence) description of the purpose of the test.

Inputs: specify each input required to execute the test. Some of the inputs will be specified by value (or perhaps a range of acceptable values), other inputs by name (e.g., the name of a file containing data to be processed).

Outputs: specify all of the outputs and features (e.g., response times) required of the test items.

Actions: describe the actions that must be carried out to execute the test.

5.3 Required Test Specifications

Test specifications shall be written by the group for subsystem testing and system testing.

The Subsystem Test Specification must be derived from the corresponding test plan and from the Design Specification.

The System Test Specification must be derived from the corresponding test plan and from the Requirements Specification and the Design Specification.

6 TEST RESULT REPORTING

Details of test results must be maintained in a Test Report folder (an A4 ring binder or similar). This must have three sections labelled 'Module Tests', 'Subsystem Tests', and 'System Tests' respectively.

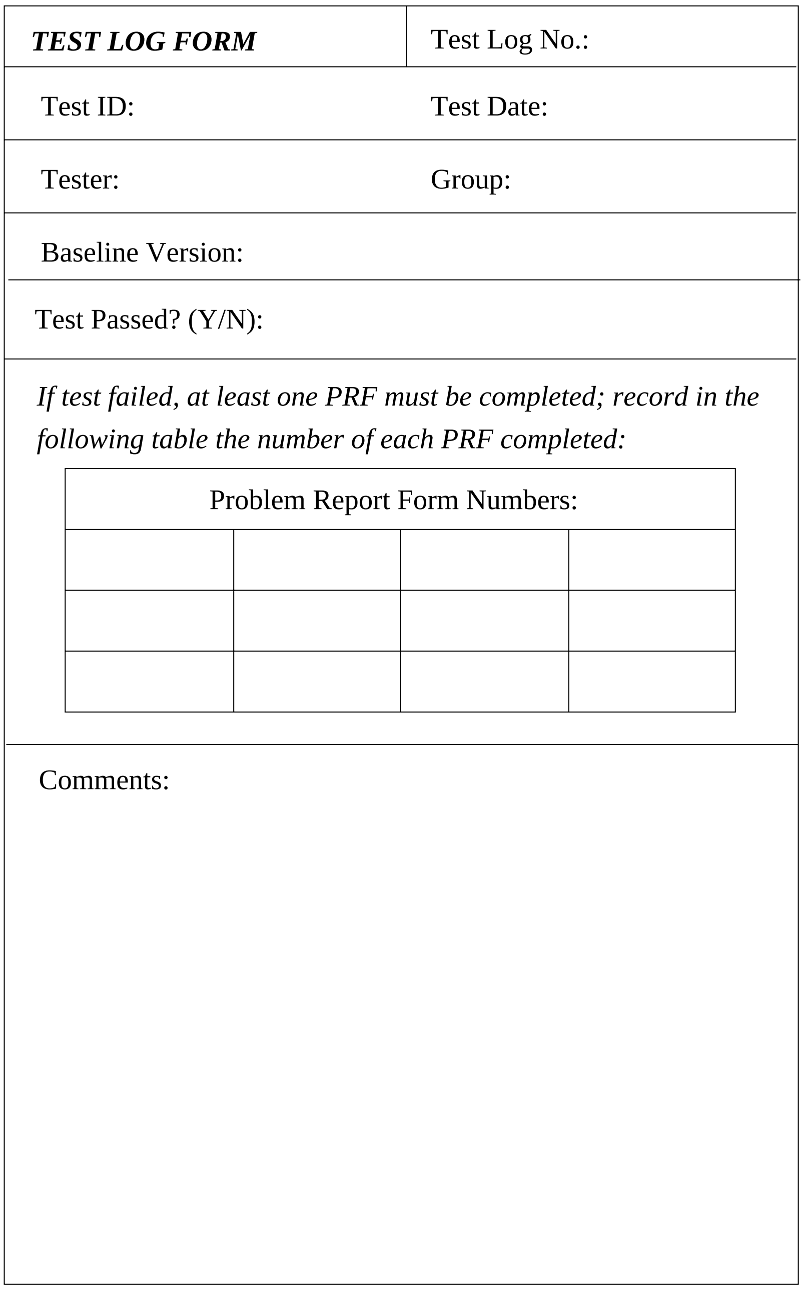
Test results must be recorded on Test Log Forms (TLFs, see Appendix A). For module testing, only one TLF is required. On successful completion of the module tests, a TLF must be completed indicating this fact and the TLF must be placed in the 'Module Tests' section of the Test Report.

For subsystem and system tests, the results of each individual test procedure must be recorded on a TLF. On completion of an individual test procedure, the corresponding completed TLF must be placed in the appropriate section of the Test Report folder. The TLFs will then form a record of the results of every test carried out.

7 BASELINES

It is most important that all items comprising a system to be tested are 'frozen' into a baseline. The documents cross-referenced in the test plans and specifications also form part of the baseline. Baselines are described in QA Document SE.QA.08 [4]. The Test Plan must indicate the version of the baseline to be used for the tests, and whilst tests are being conducted, the baseline must remain unchanged. Since it is the QA Manager who creates the baseline, it is this person's responsibility to ensure that it remains unaltered throughout a test.

A EXAMPLE TEST LOG FORM



TEST LOG FORM	Test Log No.:
Test ID:	Test Date:
Tester:	Group:
Baseline Version:	
Test Passed? (Y/N):	

If test failed, at least one PRF must be completed; record in the following table the number of each PRF completed:

Problem Report Form Numbers:			

Comments:

Figure 1: *Test log form available in /dcs/repos/Group_project/ as files* TLF.fig *(xfig format) and* TLF.eps *(encapsulated PostScript format).*

REFERENCES

[1] H.R. Nicholls. Software engineering group projects – quality assurance plan. Technical Report SE.QA.01, University of Wales, Aberystwyth, 1997.

[2] H.R. Nicholls. Software engineering group projects – project management standards. Technical Report SE.QA.02, University of Wales, Aberystwyth, 1997.

[3] H.R. Nicholls. Software engineering group projects – general documentation standards. Technical Report SE.QA.03, University of Wales, Aberystwyth, 1997.

[4] H.R. Nicholls. Software engineering group projects – operating procedures and configuration management standards. Technical Report SE.QA.08, University of Wales, Aberystwyth, 1997.

[5] I. Sommerville. *Software Engineering.* Addison-Wesley, Wokingham, 4th edition, 1992.

[6] I. Sommerville. *Software Engineering.* Addison-Wesley, Wokingham, 5th edition, 1995.

[7] S.L. Pfleeger. *Software Engineering: The Production of Quality Software.* MacMillan, New York, 2nd edition, 1991.

DOCUMENT CHANGE HISTORY

Version	*CCF No.*	*Date*	*Sections changed from previous version*	*Change d b y*
1.1	N/A	8 Dec 92	initial development	HRN
1.2	N/A	9 Dec 92	detail added to section 4	HRN
2.1	15	10 Dec 92	first draft	HRN
3.1	16	10 Dec 92	first draft	HRN
3.2	32	25 Nov 93	"csthor" changed to "thor"	FWL
			sccs style incorporated	
			corrected typography	
3.3	39	8 Oct 94	added page break after Doc. Chg. Hist	MBR
3.4		5 Oct 95	moved change history	GCC
			revised section 1	
3.5		6 Oct 97	Updated for new proj	CJP
3.6		19 Oct 98	Updated for 98	CJP
3.7		17 Oct 00	Minor changes for 2000	CJP