Testing

CxOne Standard

CxStand_Testing.doc

November 3, 2002

Construx

SOFTWARE

cxone

Advancing the Art and
Science of Commercial
Software Engineering

Contents

1 INTRODUCTION 1

1.1 OVERVIEW 1
1.2 GOALS 1
1.3 BACKGROUND 1
1.4 RELATIONSHIP TO OTHER CKAS 1
1.4.1 Testing and Quality 1
1.4.2 Testing and Engineering Management 2
1.4.3 Testing and Requirements 2
1.4.4 Testing and Design 2
1.4.5 Testing and Construction 2
1.5 RELATIONSHIP TO EXTERNAL STANDARDS 2
1.5.1 SWEBOK 2
1.5.2 IEEE Standards 2

2 ORGANIZATION 3

2.1 PLANNING 3
2.1.1 Test Planning Interactions 3
2.2 TEST TYPES 3
2.3 TEST TECHNIQUES 3

3 TEST TYPES 4

3.1 CONSTRUCTION TESTING 4
3.1.1 Single Step Test 4
3.1.2 Unit Test 4
3.1.3 Component Test 4
3.1.4 Bench Test 4
3.1.5 Developer Integration Test 5
3.2 SYSTEM TESTING 5
3.2.1 Smoke Test 5
3.2.2 Feature Test 5
3.2.3 Integration Test 5
3.2.4 Release Test 5
3.2.5 Acceptance Test 6
3.3 SPECIAL TEST TYPES 6
3.3.1 Regression Test 6
3.3.2 Performance Test 6
3.3.3 Stress Test 6
3.3.4 Configuration Test 6
3.3.5 Compatibility Test 6
3.3.6 Alpha and Beta Test 7

4 TEST TECHNIQUES 8
4.1 PREPARATION 8
4.1.1 Formal Testing 8
4.1.2 Informal Testing 8
4.2 EXECUTION 8
4.2.1 Manual Testing 8
4.2.2 Automated Testing 8
4.3 APPROACH 9
4.3.1 Structural Testing 9
4.3.2 Functional Testing 10

Copyright Notice

1 Introduction

This standard documents CxOne's definition and organization of the knowledge, artifacts, and materials for the testing Construx knowledge area (CKA).

1.1 Overview

Testing is a quality control activity that involves the dynamic execution of software systems. There is considerable industry literature available on different testing strategies, artifacts, and processes, however the terminology used can vary. This standard defines a taxonomy for discussing, organizing, and using testing concepts.

The testing CKA consists of a set of fundamentals along with support for creation artifacts to support testing. The CKA is split into test planning, test types, and test techniques.

1.2 Goals

CxOne support for testing focuses on the following goals:

- Defining a common terminology.
- Distillation of common principles into readily available checklist items.
- Provide a resource for information about existing best practices, methodologies, and techniques.
- Increase productivity by addressing issues in a uniform fashion, reducing common errors, duplication of effort, and multiple solutions for the same problem.

1.3 Background

CxOne testing materials are synthesized from many sources including the SWEBOK, IEEE software standards, Steve McConnell's works, other industry sources and literature, and Construx's experience with software projects, consulting, and training.

1.4 Relationship to other CKAs

While each CKA interacts to some level with all other CKAs, Testing interacts to the highest degree with quality, engineering management, requirements, design, and construction.

1.4.1 Testing and Quality

Testing is a subset of quality and is defined as dynamic quality control, i.e., defect detection by exercising code. The quality CKA offers additional material to provide a complete quality process.

1.4.2 Testing and Engineering Management

Projects will normally break out a separate test plan, but it should fit under the overall project plan and management processes defined in the engineering management CKA.

1.4.3 Testing and Requirements

Development of functional requirements and test cases can often occur as parallel activities, as many test cases can be derived from functional requirements, and developing test cases can strengthen the defining of functional requirements.

1.4.4 Testing and Design

Consideration of test support at design time can often allow for increased test coverage and efficiency while testing, especially with automated testing.

1.4.5 Testing and Construction

Adding support for testing during construction can create systems that are easier to test and is often critical for implementation of automated testing. Some types of testing are naturally done as part of construction activities (e.g., single-step testing, bench testing, etc.).

1.5 Relationship to External Standards

1.5.1 SWEBOK

The CxOne testing area has made the following adaptations of the SWEBOK organization:

1. Portions of *Managing the Testing Process* are dealt with in the engineering management and configuration management CKAs.
2. *Test Related Measures* are dealt with in the quality and engineering process CKAs.

These changes were made to support a more pragmatic, usable taxonomy for materials.

1.5.2 IEEE Standards

IEEE 610 defines testing as:

> "An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component"

CxOne material explicitly and implicitly supports IEEE 610, but as noted in the SWEBOK section some testing issues are supported in other CKAs for usability reasons.

IEEE 829

CxTemp_TestPlan supports reasonably straightforward conversion of content to the format of IEEE 829.

2 Organization

This section provides an overview of the different sub-areas in the CxOne testing area. Each sub-area has its own folder which contains materials supporting that area.

2.1 Planning

The amount of testing to be done is primarily a business decision based on the amount of risk the project sponsor wishes to assume and any external mandates. Testing all possible paths and permutations of a software system is a practical impossibility. Thus the planning of testing is a technical exercise to employ techniques that optimize coverage for least effort, and a business decision to determine what overall level of effort should be spent on testing.

CxOne supports material for creating test plans, designing test cases, and reporting test results.

2.1.1 Test Planning Interactions

Project Planning
Test planning is part of overall project planning. Depending on the size and nature of the project, test planning may be incorporated into the project plan or may be a separate document. The checklists and templates in this section can be used in either situation to ensure the critical items are addressed in the plan.

Quality Planning
Test planning is a subset of quality planning. Normally the quality plan will outline the role testing is to play on a project, and the test plan will cover the planning specific to testing.

2.2 Test Types

CxOne test types capture different families of testing process and context. Test types capture a combination of what, why, how, and when something is tested. Test types are combined with test techniques to create test activities that meet the unique needs of each project.

2.3 Test Techniques

Test techniques capture different strategies for testing. Test techniques cover different ways testing can be accomplished. Test techniques are combined together with each other and with test types to create test activities that meet the unique needs of each project.

3 Test Types

Test types capture a combination of what, why, how, and when something is tested. Test types are combined with test techniques to create specific test activities.

Although many test types tend to be combined with certain test techniques, there are no hard and fast rules. Some test types imply certain lifecycle stages, software deliverables, or other project context (e.g., release test). Other test types are general enough to be done almost any time on any part of the system (e.g., bench test). Some require a particular methodology (e.g., single step test). When appropriate, common utilizations of a particular test techniques will be described.

A project's test plan will normally define the test activities that will be used on the project, when they will be used, and their interactions with other project activities. Test cases are then created to support each test activity.

3.1 Construction Testing

The term construction testing is used to capture a combination of the following test types which are normally performed as part of construction activities by the engineers doing the construction.

3.1.1 Single Step Test

Single step testing is performed by stepping through new or modified statements of code with a debugger. Single step testing is often manual and informal.

3.1.2 Unit Test

Unit testing exercises a software unit independently from its system. A unit is an abstract term for the smallest thing that can be conveniently tested. This will vary based on the nature of a project and its technology but usually focuses at the class or subroutine level. Unit testing is usually automated and usually requires support of a harness, stubs, or drivers.

3.1.3 Component Test

A component is an aggregate of one or more units and components. Component testing expands unit testing to include behavior of groups, dependencies, and data integration. On some projects there will not be a meaningful distinction between unit and component tests. Component testing is often automated and may require consideration during the design phase for creation of a supporting framework, harness, stubs, and/or drivers.

3.1.4 Bench Test

Bench testing is feature or system testing performed in a development environment after a local build, usually as part of ongoing construction activities. Bench testing is often manual and informal.

3.1.5 Developer Integration Test

Developer integration testing is functional testing of a component after the component has been released and the system has been built and deployed into a standard testing environment. Special attention is given to integration of the component with the rest of the system.

3.2 System Testing

System testing occurs when all necessary components have been released internally and the system has been built and deployed into a standard environment that reasonable replicates operating conditions. System testing is concerned with the behavior of the whole system. When appropriate, system testing encompasses all external software, hardware, operating environments, etc. that will make up the final system.

The tests below are often performed by individuals or organizations that did not participate in construction of the system. System testing is commonly referred to as “QA Testing”.

3.2.1 Smoke Test

Smoke testing determines whether the system is sufficiently stable and functional to warrant the cost of further, more rigorous testing. Smoke testing may also communicate the general state of the current code base to the project team. Standards for the scope and success criteria of smoke tests may vary widely among projects. Smoke testing is most effective when automated and tied to the build process.

3.2.2 Feature Test

Feature testing is functional testing directed at specific features or subsets of system functionality, often vertical slices of the system. The features are tested for correctness and proper integration into the system. Feature testing occurs after all components of a feature have been completed and released by development.

3.2.3 Integration Test

Integration testing focuses on verifying the functionality and stability of a system or subsystem when it is integrated with internal sub-systems, external systems, third party components, or other external interfaces.

3.2.4 Release Test

Release tests ensure that release builds can be successfully deployed. This includes product deployment (web download, cd install, etc.), installation, and a pass through the primary functionality. This test is done immediately before releasing to the customer.

3.2.5 Acceptance Test

Acceptance testing compares the system to a predefined set of acceptance criteria. If the acceptance criteria are satisfied by the system, the customer will accept delivery of the system.

3.3 Special Test Types

The following test types do not fit into particular categories and may be combined with other test types.

3.3.1 Regression Test

Exercises functionality that has stabilized (i.e. have not failed for a predetermined number of testing cycles). Once high confidence has been established for certain parts of the system, it is generally wasted effort to continue rigorous, detailed testing of those parts. However, it is possible that continued evolution of the system will have negative effects on previously stable and reliable parts of the system. Regression testing offers a low-cost method of detecting such side effects. Regression testing is often automated and focused on critical functionality.

3.3.2 Performance Test

Performance testing measures the efficiency of the system with respect to time and resources under typical usage. Performance testing is often combined with instrumentation and analysis during construction (e.g., adding trace statements with time stamps).

3.3.3 Stress Test

Stress testing evaluates the performance of the test item during extreme usage patterns. Typical examples of “extreme usage patterns” are large data sets, complex calculations, extended operation, limited system resources, etc.

3.3.4 Configuration Test

Configuration testing evaluates the performance of the test item under a range of system configurations. Relevant configuration issues depend upon the particular system and may include platforms, peripherals, network patterns, configuration settings, etc. Configuration testing can also include scalability tests to ensure a system meets scalability requirements.

3.3.5 Compatibility Test

Compatibility testing is similar to configuration testing, but is focused on determining compatibility of a particular sub-system when interacting in a system with other sub-systems that can operate in that system. This is often performed on open-ended consumer systems.

3.3.6 Alpha and Beta Test

Alpha and beta testing consists of deploying the system to many external users who have agreed to provide feedback about the system. Beta testing may also provide the opportunity to explore release and deployment issues (e.g. mass replication, system versioning).

4 Test Techniques

Test techniques cover different ways testing can be accomplished. Test techniques are combined together with each other and with the test types to define test activities.

This section covers the different test techniques defined by CxOne. These techniques are often combined in standard ways with certain test types, but there are no hard and fast rules. Test activities should be optimized to the unique goals and constraints of each project.

4.1 Preparation

4.1.1 Formal Testing

Testing performed with a plan, documented set of test cases, etc that outline the methodology and test objectives. Test documentation can be developed from requirements, design, equivalence partitioning, domain coverage, error guessing, etc.

The level of formality and thoroughness of test cases will depend upon the needs of the project. Some projects can have rather informal 'formal test cases', while others will require a highly refined test process. Some projects will require light testing of nominal paths while others will need rigorous testing of exceptional cases.

4.1.2 Informal Testing

Ad hoc testing performed without a documented set of objectives or plans. Informal testing relies on the intuition and skills of the individual performing the testing. Experienced engineers can be productive in this mode by mentally performing test cases for the scenarios being exercised.

4.2 Execution

4.2.1 Manual Testing

Manual testing involves direct human interaction to exercise software functionality and note behavior and deviations from expected behavior.

4.2.2 Automated Testing

Testing that relies on a tool, built-in test harness, test framework, or other automatic mechanism to exercise software functionality, record output, and detect deviations. The test cases performed by automated testing are usually defined as software code or script that drives the automatic execution.

4.3 Approach

4.3.1 Structural Testing

Structural testing depends upon knowledge of the internal structure of the software. Structural testing is also referred to as white-box testing.

There are two types of structural testing; data flow coverage and control flow coverage. Data flow coverage tests the paths from the definition of a variable to it use. Control flow coverage tests the execution paths of the software.

The following table outlines the types of control flow coverage, listed from lowest to highest level of coverage.

Control Flow Coverage

Name	Definition
Statement Coverage	Every statement in the code under test has been executed.
Branch Coverage	Every point of entry and exit in the program has been executed at least once, and every decision in the program has taken all possible outcomes at least one.
Condition Coverage	Branch coverage with the additional requirement that “every condition in a decision in the program has taken all possible outcomes at least once.”
Multiple Condition Coverage	Multiple condition coverage requires that all possible combinations of the possible outcomes of each condition have been tested.
Modified Condition Coverage	Modified condition coverage requires that each condition has been tested independently.

4.3.2 Functional Testing

Functional testing compares the behavior of the test item to its specification without knowledge of the item's internal structure. Functional testing is also referred to as black box testing.

The following table outlines the types of functional testing coverage:

Coverage

Name	Definition
Requirements	Requires at least one test case for each specified requirement. A traceability matrix can be used to insure that requirements coverage has been satisfied.
Input Domain	Executes a function with a sufficient set of input values from the function's input domain (the set of values it can take as input parameters). The notion of a sufficient set is not completely definable, and complete coverage of the input domain is typically impossible. Therefore the input domain is broken into subsets, or **equivalence classes**, such that all values within a subset are likely to reveal the same defects. Any one value within an equivalence class can be used to represent the whole equivalence class. In addition to a generic representative, each extreme value within an equivalence class (e.g. the minimum and maximum in a numeric range) should be covered by a test case. Testing the extreme values of the equivalence classes is referred to as **boundary value** testing.
Output Domain	Executes a function in such a way that a sufficient set of output values from the function's output domain (the set of values it can return) is produced. Equivalence classes and boundary values (see Input Domain Coverage) are used to provide coverage of the output domain. A set of test cases that "reach" the boundary values and a typical value for each equivalence class is considered to have achieved output domain coverage.