NEC
Empowered by Innovation

University of Waterloo

Software Testing and Quality Assurance

Theory and Practice

Chapter 11

System Test Design

© Naik & Tripathy

Outline of the Chapter

- Test Design Factors
- Requirement Identification
- Test Objective Identification
- Modeling a Test Design Process
- Modeling Test Results
- Test Design Preparedness Metrics
- Test Case Design Effectiveness

© Naik & Tripathy

Test Design Factors

The following factors must be taken into consideration during the design of system tests

- Coverage metrics
- Effectiveness
- Productivity
- Validation
- Maintenance
- User skills

Test Case Identifier	Requirement Identifier			
	N_1	N_2	.....	N_p
T_1	A_{11}	A_{12}	.....	A_{1p}
T_2	A_{21}	A_{22}	.....	A_{2p}
T_3	A_{31}	A_{32}	.....	A_{3p}
.....	.....	.....	.....	.....
.....	.....	.....	.....	.....
T_q	A_{q1}	A_{q2}	.....	A_{qp}

Table 11.1: Coverage matrix $[A_{ij}]$.

© Naik & Tripathy

Requirement Identification

Statistical analysis conducted by Vinter reveals that out 1000 defects:

- 23.9 % requirement issues
- 24.3 % functionality
- 20.9% component structure
- 9.6 % data
- 4.3 % implementation
- 5.2 % integration
- 0.9 % architecture
- 6.9 % testing
- 4.3 % others

© Naik & Tripathy

Requirement Identification

- Requirements are a description of the needs or desire of users that a system suppose to implement
- Two major challenges in defining requirements:
  - Ensure that the right requirements are captured which is essential for meeting the expectations of the users
  - Ensure that requirements are communicated unambiguously to the developers and testers so that no surprise when the system is delivered
- Essential to have an unambiguous representation of the requirements
- The requirements must be available in a centralized place so that all the *stakeholders* have the same interpretation of the requirements

A **stakeholder** is a person or an organization who influences a system's behavior or who is impacted by the system

© Naik & Tripathy

Requirement Identification

- The state diagram of a simplified requirement life-cycle starting from *Submit* state to the *Closed* state is shown in Figure 11.1and the schema in Table 11.1
- At each of these states certain actions are taken by the owner, and the requirement is moved to the next state after the actions are completed
- A requirement may be moved to *Decline* state from any of the following state: *Open*, *Review*, *Assign*, *Implement*, and *Verification* for several reasons
- A marketing manager may decide that the implementation of a particular requirement may not generate revenue and may decline a requirement.

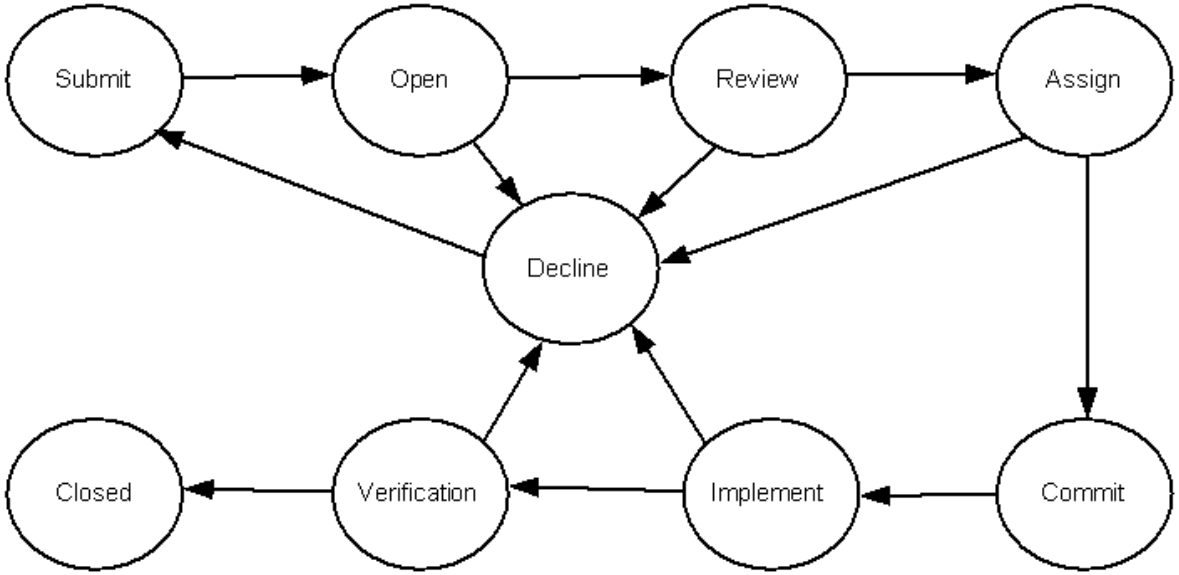

Figure 11.1: State transition diagram of a requirement

© Naik & Tripathy

Requirement Identification

Field Name	Description
requirement_id	A unique identifier associated with the requirement.
title	Title of the requirement. One line summary of the requirement.
description	Description of the requirement.
state	Current state of the requirement. It can take value from the set {Submit, Open, Review, Assign, Commit, Implement, Verification, Closed, Decline}.
product	Product name.
customer	Name of the customer who requested this requirement.
note	Submitter's note. Any additional information the submitter wants to provide that will be useful to marketing manager or director of software engineering.
software_release	Assigned to a software release. The software release number in which the requirement is desired to be available for the end customer.
committed_release	The software release number in which the requirement will be available.
priority	Priority of the requirement. It can take value from the set {high, normal}.
severity	Severity of the requirement. It can take value from the set {critical, normal}.
marketing_justification	Marketing justification for the existence of the requirement.
eng_comment	Software engineering director's comment after the review of the requirement. These comments are useful when developing functional specification, coding, or unit testing.
time_to_implement	Estimated in person - week. Time needed to implement this requirement including developing functional specification, coding, unit, and integration testing.
eng_assigned	Assigned to an engineer by the software engineering director in order to review the requirement.
functional_spec_title	Functional specification title.
functional_spec_name	Functional specification file name.
functional_spec_version	Latest version of Functional specification. The functional specification is version controlled.
decline_note	Explanation of why the requirement is declined.
ec_number	Engineering Change (EC) document number.
attachment	Attachment (if any).
tc_id	Test case identifier; multiple test case identifiers can be entered; these values may be obtained automatically from test factory database.
tc_results	Test case result; it can take value from the set {Untested, Passed, Failed, Blocked, Invalid}; these can be automatically obtained from the test factory database.
verification_method	Verification method; T - by testing; A - by analysis; D - by demonstration; I - by inspection.
verification_status	Verification state (Passed, Failed, Incomplete) of the requirement.
compliance	Compliance. It can take value from the set {compliance, partial compliance, non-compliance}.
testing_note	Notes from the test engineer. The notes may contain explanation of analysis, inspection, or demonstration given to the end customer by the test engineer.
defect_id	Defect identifier. This value can be extracted from test factory database along with the test results. If the *tc_results* field takes the value "Failed", then the defect identifier is associated with the failed test case to indicate the defect that causes the failure.

Table 11.1: Requirement schema field summary

© Naik & Tripathy

Requirement Identification

Definition of requirements traceability:

The requirements traceability is the ability to describe and follow the life of a requirement, in both forward and backward direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phase

© Naik & Tripathy

Requirement Identification

- One can generate traceability matrix from the requirement life-cycle system, which gives them confidence about test coverage

- A traceability matrix allows one to find a two-way mapping between requirements and test cases as follows
  - From a requirement to a functional specification to specific tests which exercise the requirements
  - From each test case back to the requirement and functional specifications

- A traceability matrix finds two applications:
  - To identify and track the functional coverage of a test
  - To identify which test cases must be exercised or updated when a system evolves

© Naik & Tripathy

Characteristics of Testable Requirements

- One way to determine the requirement description is testable is as follows:
- Take the requirement description **"The system must perform X."**

- Encapsulate the requirement description to create a test objective: **"Verify that the system performs X correctly."**

- Review this test objective and find out if it is possible to execute it assuming that the system and the test environment are available

- If the answer to the above question is **yes**, then the requirement description is clear and detailed for testing purpose
- Otherwise, more work needs to be done to revise or supplement the requirement description

Characteristics of Testable Requirements

- The following items must be analyzed during the review of requirements:
  - Safety
  - Security
  - Completeness
  - Correctness
  - Consistency
  - Clarity
  - Relevance
  - Feasibility
  - Verifiable
  - Traceable

© Naik & Tripathy

Characteristics of Testable Requirements

- A ***functional specification*** provides
  - a precise description of the major functions the system must fulfill the requirements
  - explanation of the technological risks involved
  - external interfaces with other software modules
  - data flow such as flowcharts, transaction sequence diagrams, and finite-state machines describing the sequence of activities
  - fault handling, memory utilization and performance estimates
  - any engineering limitation

© Naik & Tripathy

Characteristics of Testable Requirements

- The following are the objectives that are kept in mind while reviewing a functional specification:
  - Achieving requirements
  - Correctness
  - Extensible
  - Comprehensive
  - Necessity
  - Implementable
  - Efficient
  - Simplicity
  - Consistency with existing components
  - Limitations

© Naik & Tripathy

Characteristics of Testable Requirements

Purpose, goals, and exception are clearly stated. Address the right objectives.
Contain the requirements and standards with which this document complies.
Clearly stated the operating environment. For what hardware, OS, and software release the feature is targeted. Minimum hardware configuration that supports this application.
Clearly list the major functions which the system must performs.
Clearly define the success criteria which the system must fulfill to be effective.
Provide understandable, organized, and maintainable model of the processes, and or data/or objects, using standard structured method and the principle of functional decomposition.
Use standard and clearly defined terminology (key words, glossary, syntax and semantics).
Display a heavy use of model and graphics (e.g., SDL-GR, Finite State Model), not primarily English narrative.
Document the assumptions.
The document should have a natural structure/flow and with each atomic feature labeled with an identifier for easy cross-referring to specific test cases.
Should have standard exception handling procedure, consistent error messages, and on-line help functions.
External interfaces, such as CLI, MIBs, EMS, and Web interface are defined clearly.
Clearly stated possible tradeoffs between speed, time, cost and portability.
Performance requirements are defined, usually in terms of packets per second, transactions per second, recovery time, response time or other such metrics.
Scaling limits and resource utilization (cpu utilization, memory utilization) are stated precisely.
Documentation of unit tests.

Table 11.4: Characteristics of testable functional specification.

Test Objective Identification

- The question **"What do I test?"** must be answered with another question: **"What do I expect the system to do?"**
- The first step in identifying the test objective is to read, understand, and analyze the functional specification
- It is essential to have a background familiarity with the subject area, the goals of the system, business processes, and system users, for a successful analysis
- One must critically analyze requirements to extract the ***inferred requirements*** that are embedded in the requirements
- An ***inferred requirement*** is one that a system is expected to support, but not explicitly stated
- ***Inferred requirements*** need to be tested just like the explicitly stated requirements

© Naik & Tripathy

Test Objective Identification

- The test objectives are put together to form a test group or a subgroup after they have been identified
- A set of (sub) groups of test cases are logically combined to form a larger group
- A hierarchical structure of test groups as shown in Figure 11.2 is called a test suite
- It is necessary to identify the test groups based on test categories, and refine the test groups into sets of test objectives
- Individual test cases are created for each test objective within the subgroups
- Test groups may be nested to an arbitrary depth
- The test grouping may be used to aid system test planning and execution that are discussed in Chapters 12 and 13, respectively

Test Objective Identification

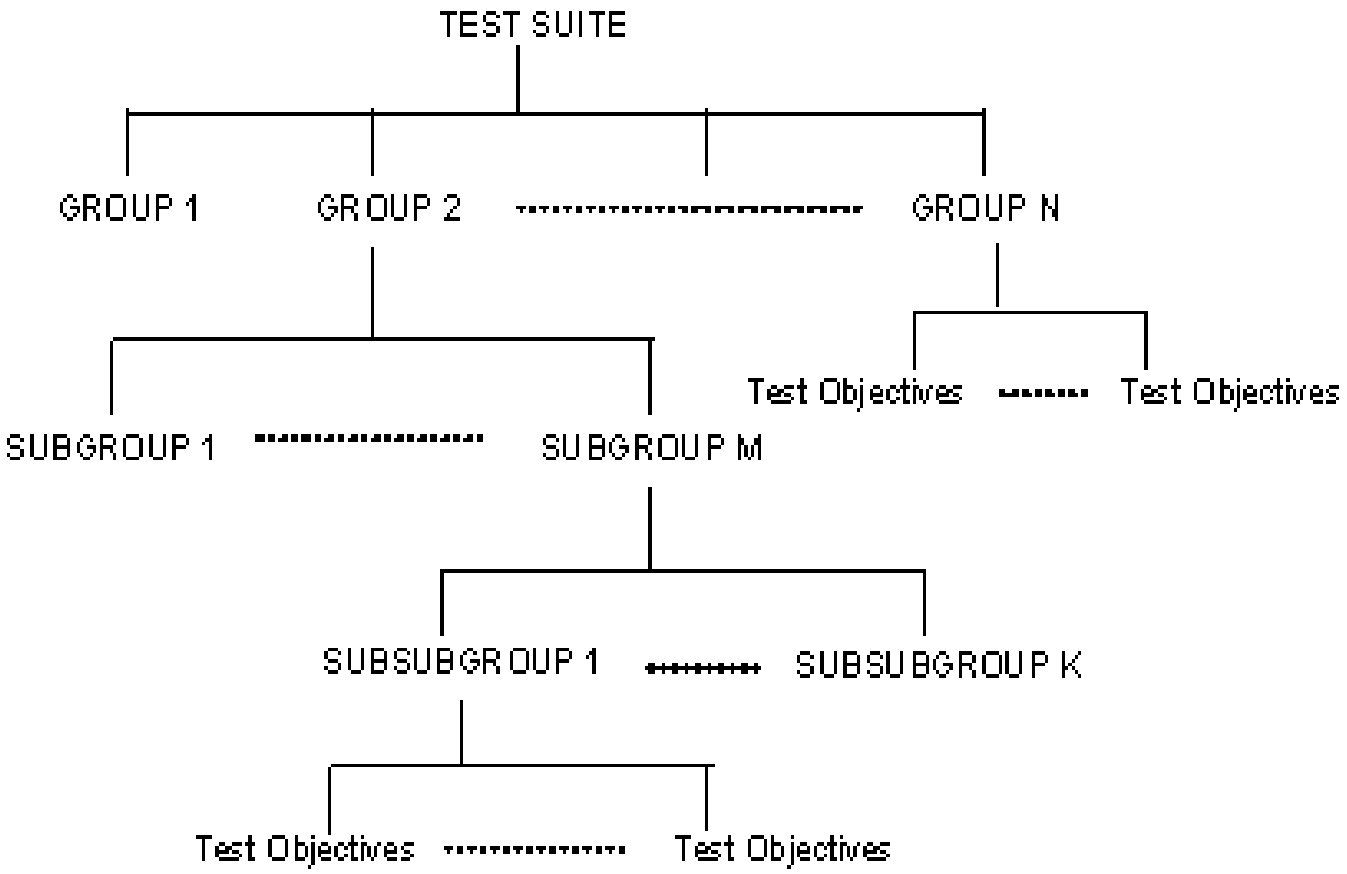

Figure 11.2: Test Suite Structure

© Naik & Tripathy

Modeling a Test Design Process

- One test case is created for each test objective
- Each test case is designed as a combination of modular components called test steps
- Test cases are clearly specified so that testers can quickly understand, borrow, and re-use the test cases
- Figure 11.6 illustrate the life-cycle model of a test case in the form of a state transition diagram
- One can easily implement a database of test cases using the test cases schema shown in Table 11.6

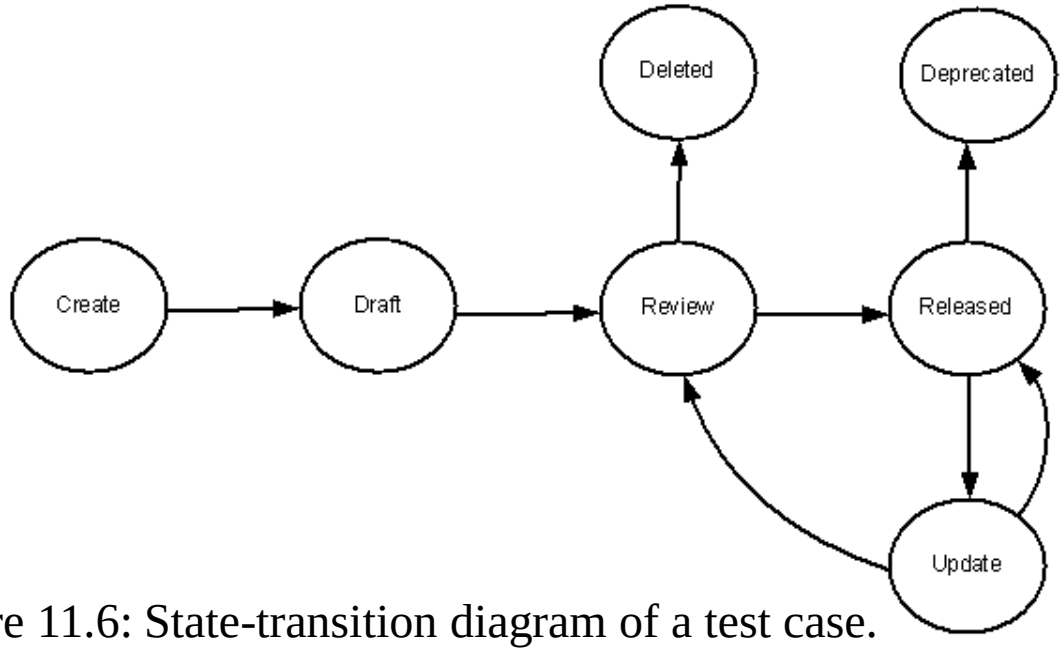

Figure 11.6: State-transition diagram of a test case.

Modeling a Test Design Process

Field	Description
tc_id	A test case identifier assigned by the test author (80 chars).
tc_title	The title of the test case (120 chars).
creator	Name of the person who created this test case.
status	Current state of the record: Create, Draft, Review, Released, Update, Deleted, and Deprecated.
owner	Current owner of this test case.
eng_assigned	Test Engineer assigned to write the test procedure.
objective	The objective of the test case (multi line string).
tc_author	The name of the test case author (user name).
originator_group	The group that originates the test (performance testing group, functional testing group, scaling testing group, etc.).
test_category	Test category name (performance, stress, inter-operability, functionality, etc.).
setup	A list of the steps to perform prior to the test.
test_steps	A list of the test steps.
cleanup	A list of post tests activities.
pf_criteria	A list of Pass/Fail criteria.
requirement_ids	A list of references to the requirements id from requirement database.
candidate_for_automation	If the test can be/should be automated.
automation_priority	It is automation priority.
review_action	Action items from the review meeting minutes.
approver_names	List of approver names.

Table 11.6: Test case schema summary.

© Naik & Tripathy

Modeling Test Results

- A test suite schema shown in Table 11.7 can be used for testing a particular release
- The schema requires a test suite id, a title, an objective and a list of test cases to be managed by the test suite
- The idea is to gather a selected number of released test cases and repackage them to form a test suite for a new project
- The results of executing those test cases are recorded in a database for gathering and analyzing test metrics
- The result of test execution is modeled by using a state-transition diagram as shown in Figure 11.7
- The corresponding schema is given in Table 11.8

© Naik & Tripathy

Modeling Test Results

Field	Description
test_suite_id	A unique identifier assigned by the originator.
test_suite_title	The test suite title – a one line title for the test suite.
test_suite_objective	The objective of the test suite – a short description.
tests	A reference list of test case identifiers.
test_id	The test case id - selected.
test_title	The test case title – read only – filled in when test case is created.
test_category	Test category name (performance, stress, inter-operability, functionality, etc.).
tester	The engineer responsible for testing.
sw_dev	The software developer responsible for this test case who will assists the test engineer in the execution of this test case.
priority	The priority of the test case.
requirement_ids	Requirement identifier – read only – filled in when test case is created.
cycle 1 - 3	Check box to indicate the test is a cycle 1, 2, or 3 test case.
regression	Check box to indicate the test is a regression test case.

Table 11.7: Test suite field summary.

Modeling Test Results

Field	Description
tc_id	A reference to a test case identifier record.
test_title	The test case title – read only – filled in when result is created.
test_category	Test category name (performance, stress, inter-operability, functionality, etc.).
status	The state of the test case result: Passed, Failed, Blocked, Invalid or Untested. Initially the status of the test case is "Untested".
run_date	The date the test case was run.
time	Time spend in executing this test case.
tester	The name of the person who ran the test.
release	The software release (03.00.00).
build	The software integration number (1-100).
defect_ids	A list to defects, which cause this test to fail. This value can come from bug tracking database.
test_suite	The test suite this results pertains to.

Table 11.8: Test result schema summary.

© Naik & Tripathy

Modeling Test Results

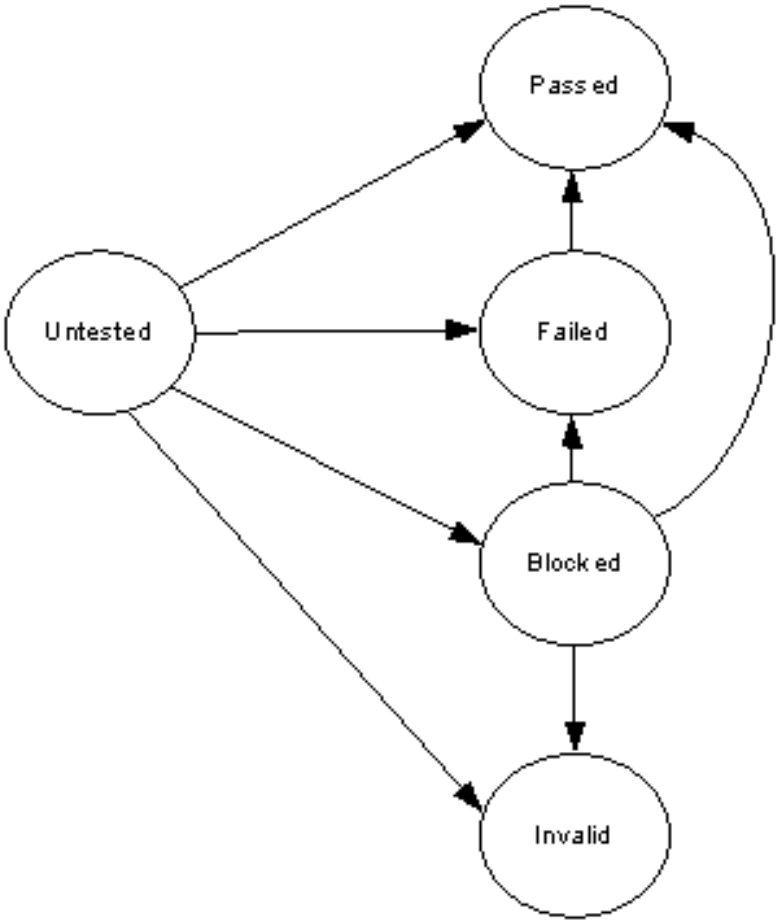

Figure 11.7: State transition diagram of a test case results

Test Design Preparedness Metrics

- Metrics are tracked
  - To know if a test project is progressing according to schedule
  - Plan the next project more accurately

- The following metrics are monitored
  - Preparation status of test cases (PST)
  - Average time spent (ATS) in test case design
  - Number of available test (NAT) cases
  - Number of planned test (NPT) cases
  - Coverage of a test suite (CTS)

© Naik & Tripathy

Test Case Design Effectiveness

- The objective of the test design effectiveness metric is to
  - Measure the *defect revealing ability* of the test suite
  - Use the metric to improve the test design process
- A metric commonly used in the industry to measure test case design effectiveness is the Test Case Design Yield (TCDY)

$$\text{TDCY} = \frac{\text{NPT}}{\text{NPT} + \text{Number of TCE}} \times 100\%$$

- TCE stand for Test Case Escaped
- During testing new defects are found for which no test cases had been planned
- For these new defects, new test cases are designed, which is known as Test Case Escaped metric