CHAPTER – 11
SYSTEM TEST DESIGN

OUTLINE OF THE CHAPTER

Test Design Factors

Requirement Identification

Test Objective Identification

Modeling a Test Design Process

Modeling Test Results

Test Design Preparedness Metrics

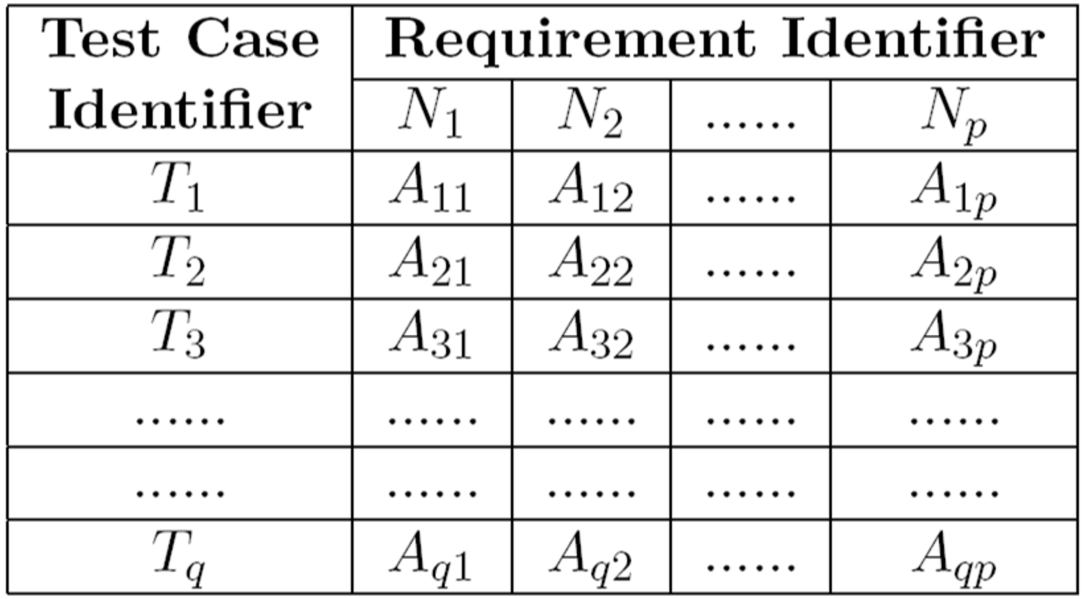
Test Case Design Effectiveness

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

A coverage matrix is used to trace the requirements from the client to the tests that are needed to verify whether the requirements are fulfilled.



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 $[A_{ij}]$.

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

REQUIREMENT IDENTIFICATION

Requirements are a description of the needs or desire of users that a system is supposed 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

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

REQUIREMENT IDENTIFICATION

The state diagram of a simplified requirement life-cycle starting from *Submit* state to the *Closed* state is shown in the Figure.

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.

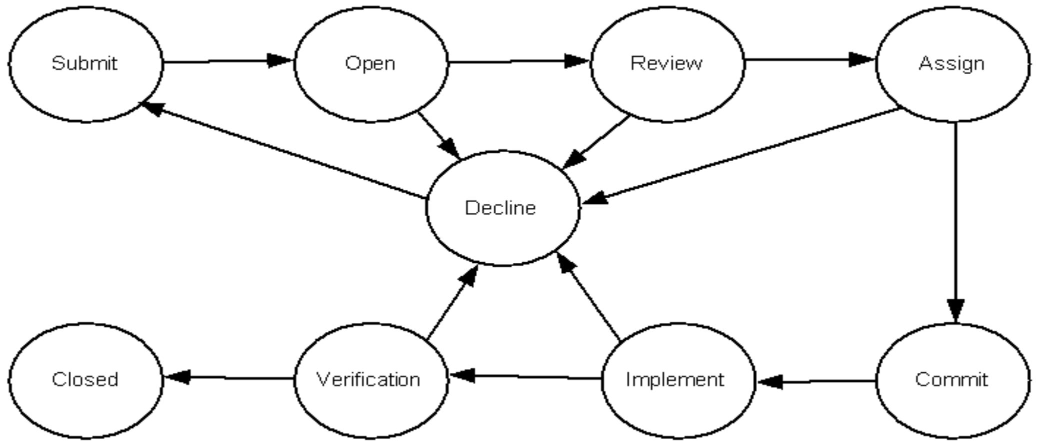
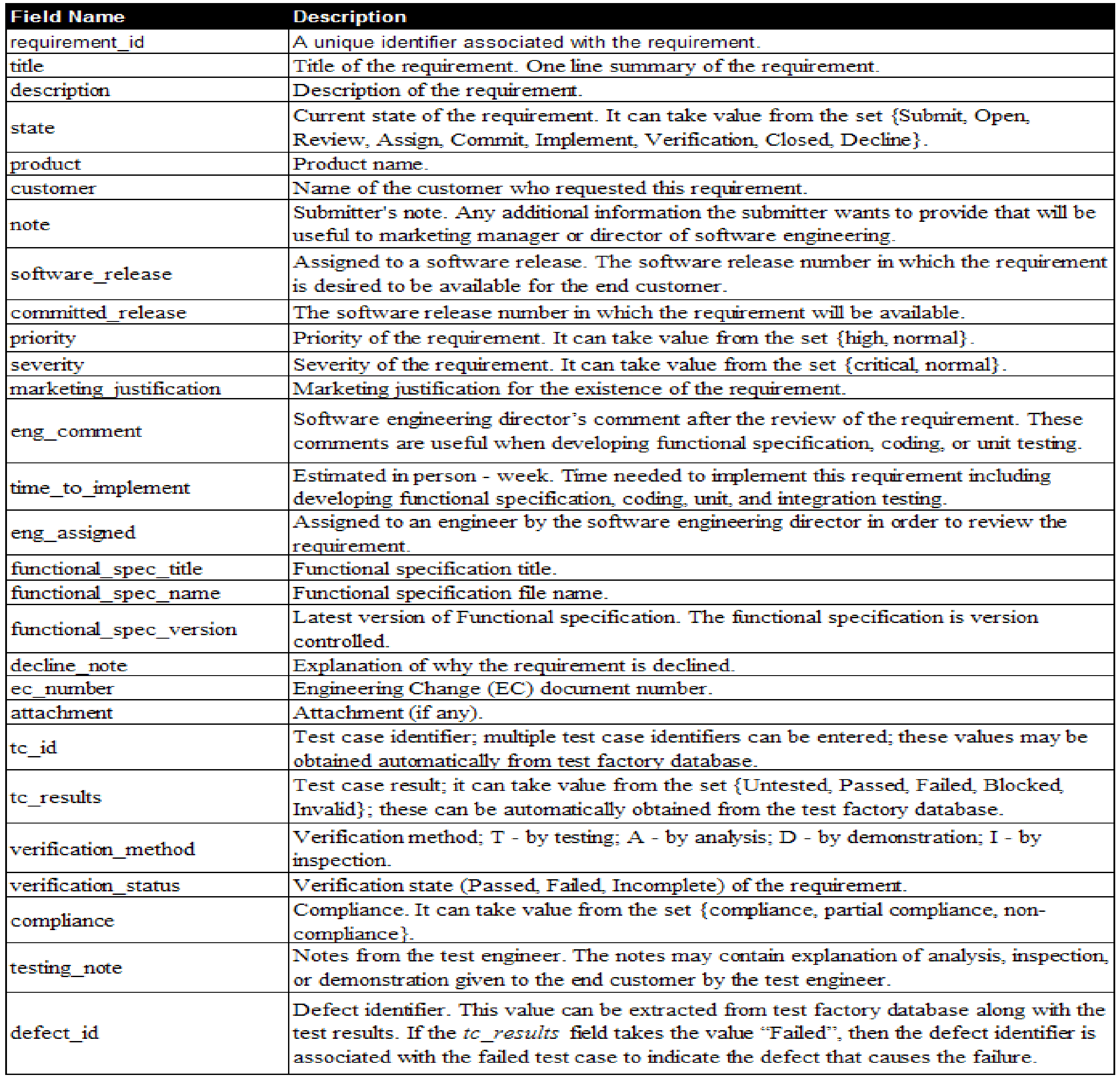


Figure 11.1: State transition diagram of a requirement

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

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 phases.

TRACEABILITY MATRIX

A traceability matrix allows one to find a two-way mapping between requirements and test cases as follows

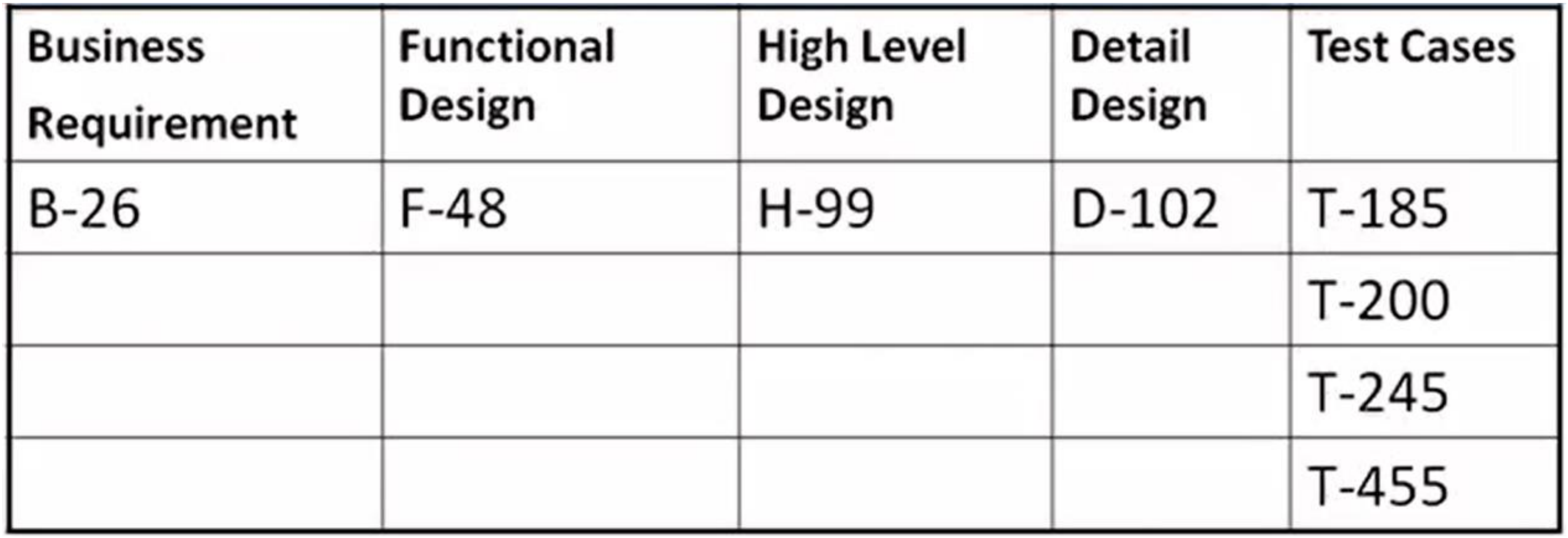
- From a requirement to a functional specification to specific tests which exercise the requirements
- From each test case back to the requirement and functional specifications

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

https://www.guru99.com/traceability-matrix.html

TRACEABILITY MATRIX



Business Requirement	Functional Design	High Level Design	Detail Design	Test Cases
B-26	F-48	H-99	D-102	T-185
				T-200
				T-245
				T-455

DIFFERENCE BETWEEN COVERAGE AND TRACEABILITY MATRIX

Coverage metrics are concerned with the extent to which the device under test (DUT) is examined by a test suite that meets certain criteria. The criteria may be node testing, branch testing, or a feature identified from a requirement specification.

A traceability matrix allows one to find a mapping between the requirements and test cases both ways as follows:
• From a requirement to a functional specification to specific tests which exercise them.
• From each test case back to the requirement and functional specification.

A traceability matrix assists is identifying and tracking the functional coverage of a test suite. Without a traceability matrix it is difficult to determine the extent of requirement coverage achieved by a particular test suite.

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

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

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

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

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

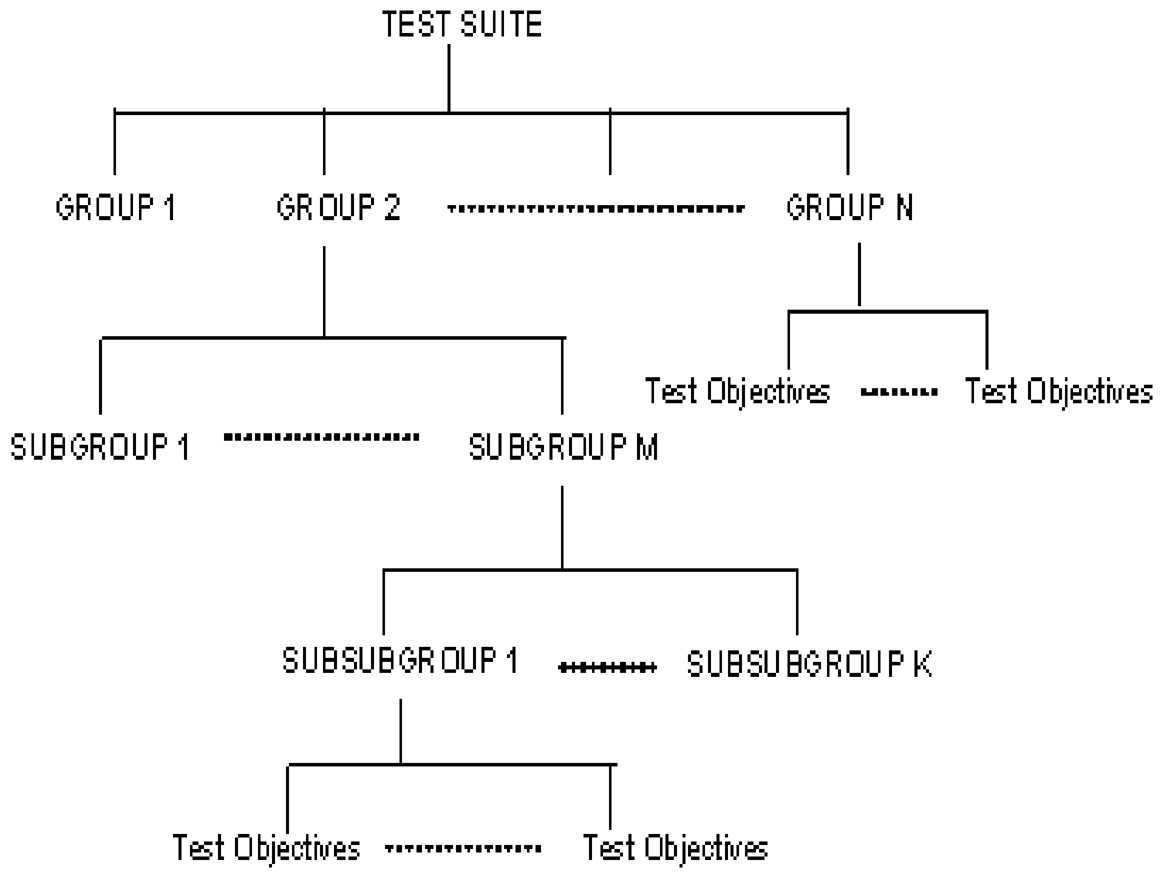


Figure 11.2: Test Suite Structure

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

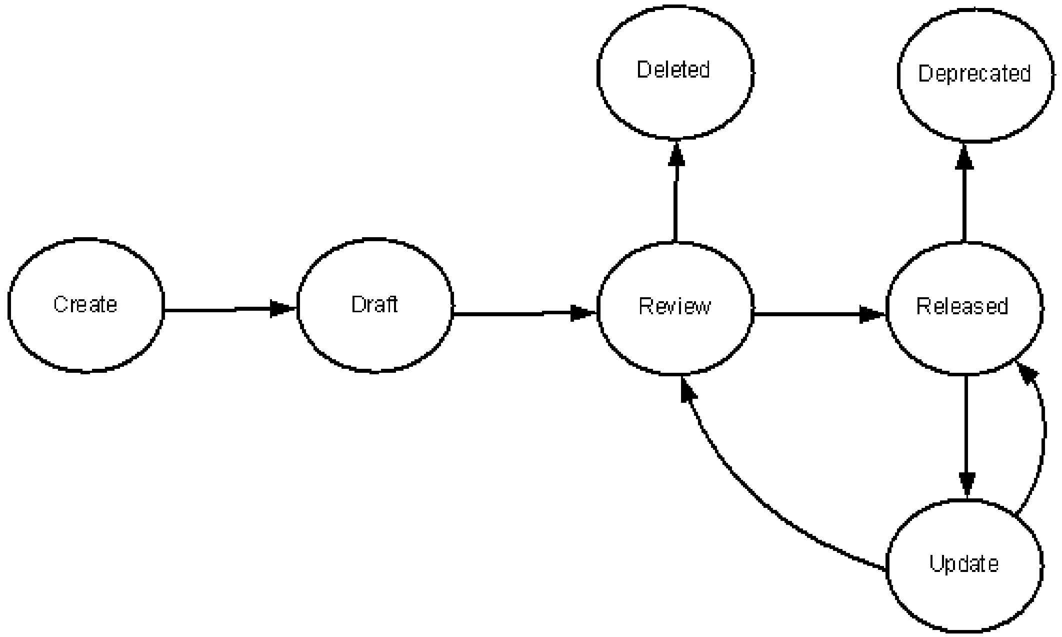


Figure 11.6: State-transition diagram of a test case.

MODELING 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

MODELING TEST RESULTS

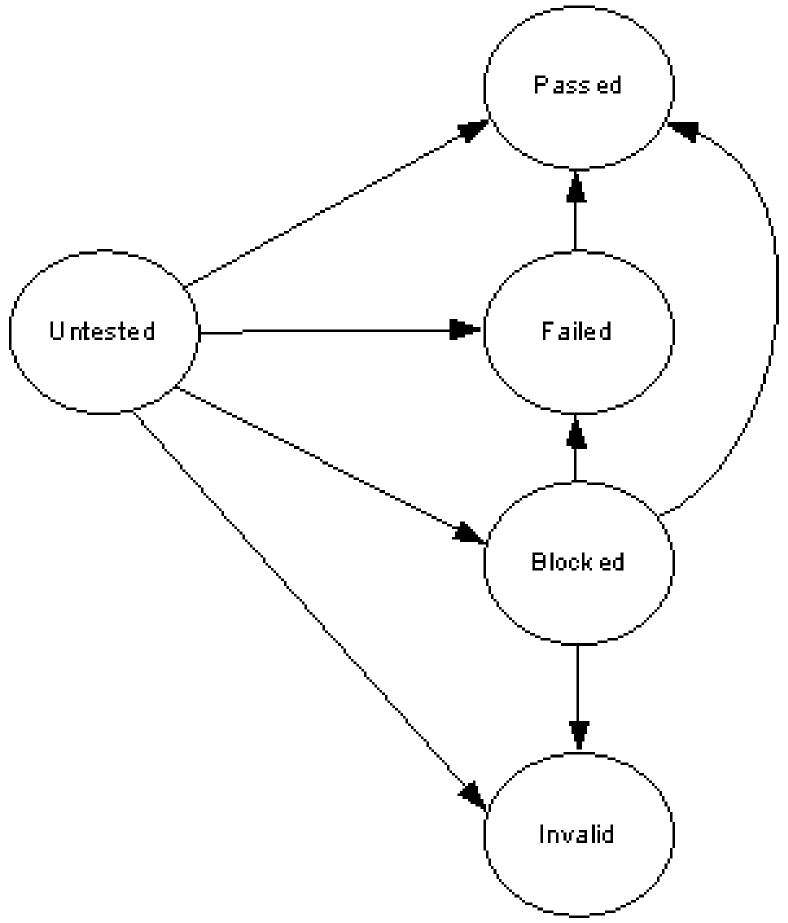


Figure 11.7: State transition diagram of a test case results

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)

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

During system level testing, defects are revealed due to the execution of planned test cases.

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 stands 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

THANK YOU!!