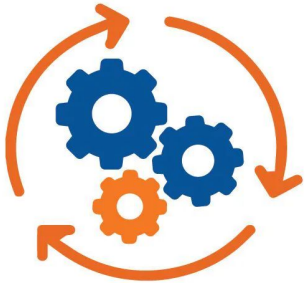
Chapter 7

Test Analysis and Design



Lecturer: Nguyen Thanh Quan (MSc)
Email: tg_nguyenthanhquan_cntt@tdtu.edu.vn

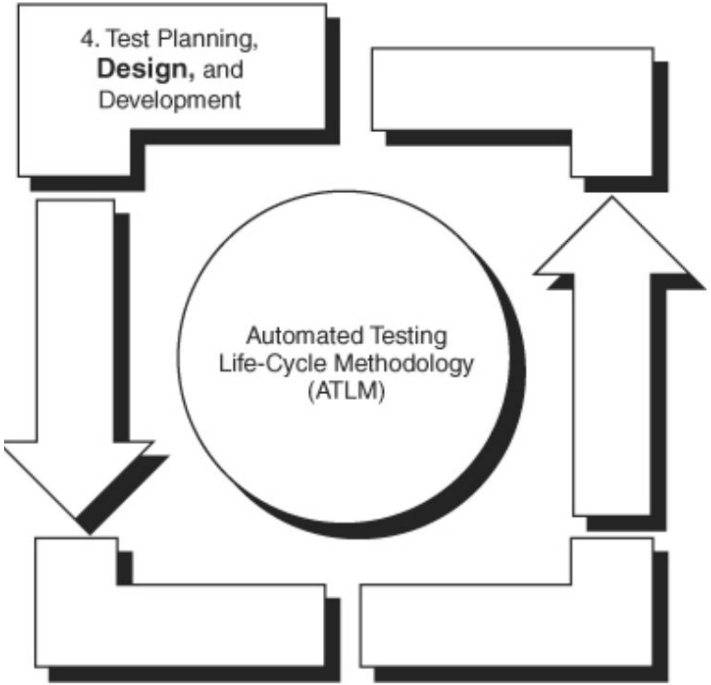
Content

1. Test Requirements Analysis
2. Test Program Design
3. Test Procedure Design

Introduction

Automated testing involves a mini-development life cycle

- An effective test program that incorporates the automation of software testing has a development life cycle of its own.
- The test development effort requires careful analysis and design.
- The test effort can be classified into two primary categories: **static** (Chapter 4 includes various reviews, inspections, and walkthroughs) **and dynamic testing** (Chapter 7 - its definition and the associated requirements and/or use case analysis and design required)



Introduction

- Test requirements statements should clearly outline test conditions that will provide the highest probability of finding errors.
- Test requirements analysis also involves the review of the system's most critical success functions and high-risk functionality, as part of risk management.
- Test requirements statements should specify attributes of the most critical success functions and high-risk functionality, and testing should then verify that those requirements have been met.

Introduction

Dynamic testing consist of

- The implementation of test techniques that involve the development and execution of test procedures designed to validate requirements.
- Verification methods are employed in the requirements validation process.

Introduction

Dynamic testing consist of

In this chapter, the following points will be discussed:

=> Methods for modeling the design of the (dynamic) test program are depicted.

=> The need to analyze whether a test procedure should be performed manually or via an automated test tool, as well as the need to associate test data requirements with test procedures.

=> Test procedure standardization and test procedure management.

Introduction

Test Analysis and Design Process

Step	Description
	Analysis
1	Goals and objectives. The test team reviews the test goals, objectives, and strategies.
2	Verification methods. Verification methods are assigned to system requirements or use cases and documented within a requirements traceability matrix.
3	Test requirement analysis. Test requirements statement definition is performed. Test requirements are derived from various system requirements of the application-under-test (AUT).
4	Test requirements matrix. Test requirements statements are mapped to system requirements or use cases and/or to system design (architecture) components.
5	Test technique mapping. A preliminary association of test requirements statements to test techniques is made. The test requirements matrix is modified to reflect this mapping.

Introduction

Test Analysis and Design Process

	Design
6	Test program model definition. Various test techniques are reviewed for their applicability to the test program, and the test team ensures that the test techniques are justified by associated test requirements. The test requirements matrix is updated accordingly. The test program model is defined to identify the test techniques that apply to the test program.
7	Test architecture definition. This activity involves the selection of a test architecture model and population of model attributes.
8	Test procedure definition. Logical groups of test procedures are defined. A naming convention for the suite of test procedures is defined. The test procedure format is defined.
9	Test procedure mapping. An association between test procedures and test requirements statements is made. The test procedure matrix is created, reflecting this mapping.
10	Automated/manual test mapping (decision what to automate). The test procedure matrix is modified. This matrix specifies whether an automated tool will support the test procedure execution, or whether the procedure will be performed manually. An additional column in the matrix identifies potential automated test script reuse assets.
11	Test data mapping. The test procedure matrix is modified to reflect test data requirements for each test procedure.

Test Requirements Analysis

Test Requirements Analysis

Analysis is aimed at identifying the different kinds of tests needed to verify the system, can be undertaken by studying the system from several perspectives, depending on the testing phase.

- **The structural approach:**
  - includes the study of the system design or the functional process flow inherent in user operations.
  - relies on unit and integration tests, also referred to as white-box tests.
  - primarily based upon review of design specifications at the development.

Test Requirements Analysis

- **The requirements or behavioral approach:**
  - referred to as the system test level, most often involves black-box testing.
  - consists of system and acceptance tests.

An alternative way of studying the system is to review critical success and highest-risk functions.

=> ensure that the test team will **adequately exercise** these functions.

=> enables the test team to design **the next progressive step** in the test analysis and design process by specifying a detailed outline of what is to be tested.

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)

An analysis should be performed to identify the test requirements that pertain to the development test effort.

- requires the review of the **detailed design and/or the software code.**
- **emphasizes** software module i**nput and output.**
- addresses test requirements from a **white-box-based view.**

The resulting test requirements statements are based on examination of the **logic** of the design or the **implementation** of the software code, where a detailed design is not available.

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)

Three different approaches for design-based test requirements analysis:

- **Statement coverage:** invokes every statement in a program at least once.
- **Decision coverage:** every point of entry and exit in the program is invoked at least once and every decision in the program takes on all possible outcomes at least once.
- **Modified condition decision coverage (MC/DC):** requires each condition (term) within a decision to be demonstrated by execution, ensuring that it independently and correctly affects the outcome of the decision.

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)

Strength:

- provide step-by-step instructions that are detailed down to keystroke entry.
- investigates whether test procedures need to be developed for both source and object code.
- find errors introduced during design, coding, compiling, linking, and loading operations.

Weaknesses:

- requires that all entries and exits of the software units be tested.
- require a large number of test procedures, consume a great deal of time, and require the work of a larger number of personnel.

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)

- The goal of these test requirements is to **uncover errors** by forcing the software to operate under **unintended or abnormal conditions.**
- The test team might also derive test requirements by analyzing the program **logic** pertaining to **decision trees**.
- The test engineer can examine the **code entry and exit conditions** and derive test requirements intended to ensure that every point of entry and exit in the program is i**nvoked at least once.**

Test requirements can include statements pertaining to testing every condition for a decision within a software program.

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)

Development Test Requirements Matrix:

- The test requirements for development-level testing should be defined and entered into a requirements **traceability** database or **matrix**.
- Each test requirement should be **associated with a system architecture** component or a design component identification number.
- Requirements management allows the test engineer to maintain these **traceability** matrices in an automated fashion.

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)/Development Test Requirements Matrix

SR ID	SWR ID	Architecture Component	TR ID	TR Statement	Test Technique
3.2.1a	SM001	Systems Management	1001	System connectivity checks shall be selectable within a range of 1 to 60, and shall not accept values outside of this range	Fault Insertion
3.2.1a	SM002	Systems Management	1002	System shall provide appropriate error message for value selections outside of range	Error Handling
3.2.1b	SM003	Systems Management	1003	System shall perform system load checks between established servers and automatically adjust loads between servers, as necessary	String Test

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)/Development Test Requirements Matrix

3.2.1c	SM004	Systems Management	1004	System shall maintain a 50% memory reserve capacity for routine load levels	Memory Usage
3.2.1d	SM005	Systems Management	1005	System shall perform daily checks to monitor user authentication/passwords	String Test
3.2.1d	SM006	Systems Management	1006	System shall note authentication/password violations	Error Handling

Test Requirements Analysis

Development-Level Test Analysis (Structural Approach)/Development Test Requirements Matrix

3.2.1d	SM007	Systems Management	1007	Users must verify new passwords twice before the new passwords will be accepted; when an error occurs in the password entry, the user shall receive an error message	Error Handling
Various	Various	Various (derived from analysis)	1008	All statements in this module should be executed at least once	Statement Coverage
Various	Various	Various (derived from analysis)	1009	Test the number of times the logical branches have been exercised for both true and false conditions	Branch Coverage

Test Requirements Analysis

System-Level Test Analysis (Behavioral Approach)

- Test analysis also needs to be performed to identify the test requirements pertaining to the system test effort.
- Test requirements analysis at the system test level is also known as the behavioral approach.
- **Requires**: the review of system/software requirement specifications.
- **Emphasizes**: on test input and expected output.
- The results are based on examination of the system/software requirements specifications.
- Tests requirements from a **black-box-based perspective.**

Test Requirements Analysis

System-Level Test Analysis (Behavioral Approach)

Another approach for deriving test requirements includes the use of f**unctional threads.**

=> Result from analysis of the functional thread of the high-level business requirements.

The test engineer reviews the high-level business process by examining the results of enterprise business process reengineering or by employing **use case analysis.**

Test Requirements Analysis

System-Level Test Analysis (Behavioral Approach)

Business process reengineering (BPR):

- is a **structured method** for analyzing the procedures.
- accomplishes goals and redesigning the business's processes with a focus on **the end user.**
- BPR examines the **entire scope of the processes.**
- Determining the flow of information through components of the business and **suggesting major improvements.**

Test Requirements Analysis

System-Level Test Analysis (Behavioral Approach)

Use case analysis:

- is a way of **modeling** requirements.
- Each use case has a defined **starting point, set of discrete steps**, and **defined exit criteria.**
- The use case construct defines the **behavior** of a system or other **semantic entity without revealing its internal structure.**
- Each use case specifies **a sequence of actions, including variants,** that the entity can perform by **interacting with actors of the entity.**

Test Requirements Analysis

System-Level Test Analysis (Behavioral Approach)

Use case analysis includes:

- the identification of what the user does.
- what the software does.
- what the user interface does.
- what the database needs to do to support the step.

=> The **accumulation** of the various user actions facilitates the development of training and user documentation as well as the design of test procedures.

=> The review of **critical success** and **high-risk functions** of the system.

Test Requirements Analysis

Critical/High-Risk Functions

1/ Provide the **highest probability** of finding errors.
2/ Generate tests that utilize **out-of-range input data**.
3/ Uncover errors by forcing the software to exercise **unintended** or **abnormal** conditions.

Rank	Function	Software Component	Indicator
1	Verify identification of trading partner account prior to any automated exchange of asset trading information	SG-07	High Risk
2	Sort through asset trade opportunities and identify the best-value trade and close deal on best-value trade	AT-12	Critical
3	Provide communications and flow of information between software components operating at different levels of security classification	SG-07	High Risk
4	Monitor exchange rates and primary economic indicators for changes in the securities market and the worldwide economy	DS-13	High Risk
5	Monitor securities and the most significant securities movement	TV-10	Critical
6	Create simulation modeling producing extended forecasts, analyze future of evolving trends, and provide long-term executive decision support	DS-13	Critical

Test Requirements Analysis

System-Level Test Analysis (Behavioral Approach)
System Test Requirements Matrix

SR ID	Architecture Component	TR ID	Test Requirement Statement	Test Technique
3.2.1a	Systems Management	2001	System shall verify the connectivity with external data sources at least once per minute	Functional Test
	Systems Management	2002	System connectivity checks verifying external data sources shall be selectable within a range of 1 to 60 seconds	Functional Test
	Systems Management	2003	System connectivity checks shall be selectable within a range of 1 to 60 seconds and must not accept values outside of this range; system shall provide an appropriate error message for value selections outside of range	Boundary Value

Test Requirements Analysis

System-Level Test Analysis (Behavioral Approach)
System Test Requirements Matrix

3.2.1b	Systems Management	2004	System shall perform system load checks between established servers and automatically adjust loads between servers	Functional Test
3.2.1c	Systems Management	2005	Performance measures need to be assessed at 4-, 16-, 24- and 48-hour intervals	Performance Test
3.2.1d	Systems Management	2006	System shall perform checks of user authentication password values against dictionaries and tables of easily guessed passwords	Functional Test
3.2.1d	Systems Management	2007	Where authentication password violations are detected, system shall prompt user to change password at the time of the next system logon	Functional Test
3.2.1d	Systems Management	2008	Users must verify their new passwords twice before they will be accepted; when error occurs in password entry, the user shall receive an error message	Functional Test

Test Program Design

Test Program Design

- **The test program** must be **mapped** out and consciously **designed** to ensure that the test **activities** performed represent the most **efficient** and **effective** tests for the system.
- Test program **resources are limited**, yet **ways of testing the system** are **endless**.

Test Program Design

Technical Environment

The results of test analysis include:

- definition of test goals and objectives.
- the selection of verification methods.
- their mapping to system requirements or use case.
- creation of test requirements statements.

=> The test requirements are then mapped to system requirements or use cases and/or system design components.

=> After this mapping is complete, a preliminary association of test requirements to test techniques is developed.

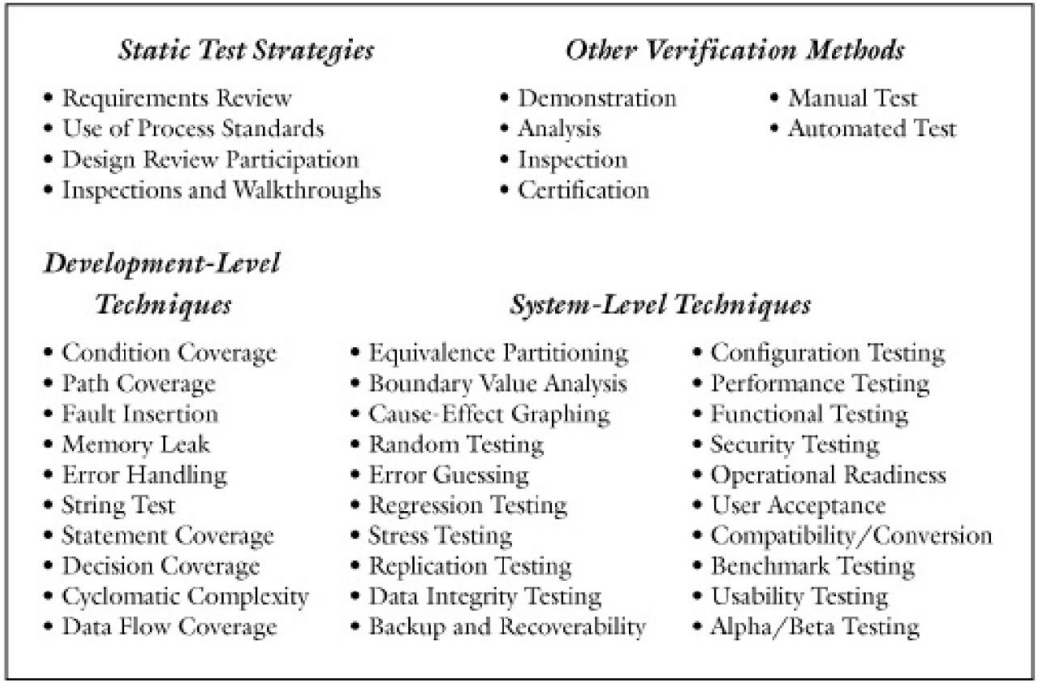
Test Program Design

Test Program Design Models

- The first of the design models consists of a **test program model.**
- The test program model consists of a graphic illustration that depicts the **scope** of the test program.
- Includes test **techniques**.
- The model may also **outline static test strategies** that are utilized throughout the application development life cycle.
- Should explicitly identify **verification methods** other than testing that will be employed during development and system-level tests.

Test Program Design

Test Program Design Models



Static Test Strategies

- Requirements Review
- Use of Process Standards
- Design Review Participation
- Inspections and Walkthroughs

Other Verification Methods

- Demonstration
- Analysis
- Inspection
- Certification
- Manual Test
- Automated Test

Development-Level Techniques

- Condition Coverage
- Path Coverage
- Fault Insertion
- Memory Leak
- Error Handling
- String Test
- Statement Coverage
- Decision Coverage
- Cyclomatic Complexity
- Data Flow Coverage

System-Level Techniques

- Equivalence Partitioning
- Boundary Value Analysis
- Cause-Effect Graphing
- Random Testing
- Error Guessing
- Regression Testing
- Stress Testing
- Replication Testing
- Data Integrity Testing
- Backup and Recoverability
- Configuration Testing
- Performance Testing
- Functional Testing
- Security Testing
- Operational Readiness
- User Acceptance
- Compatibility/Conversion
- Benchmark Testing
- Usability Testing
- Alpha/Beta Testing

Test Program Design

Test Program Design Models

- After defining a test program model, the test team's next task is to **construct** a test **architecture** which consists of a **graphic illustration.**
- The objective is to define the way that test procedures will be organized **within the test effort.**
- 2 ways of structuring:
  - One involves the **logical grouping** of test procedures.
  - One **associates test procedures** with the various kinds of test techniques.

Test Program Design

Test Program Design Models / Design-Based Test Architecture

- The design-based test architecture associates test procedures with the **hardware and software** design components of the system application.
- The logic of this model stems from the **understanding** that the **hardware** and **software** design components can be traced to system and software requirements specifications.
- The test team can refer to a requirements **traceability matrix.**

Test Program Design

Test Program Design Models / Design-Based Test Architecture

Design-Based Test Architecture

=> a clear picture of the techniques.

=> define all corresponding test procedures.

Development Test Level

SM-06	SG-07	SA-08	TV-10	Others . . .
Error Handling	Error Handling	Cyclomatic Complexity	Cyclomatic Complexity	. . .
Memory Leak	Memory Leak	Path Coverage	Path Coverage	. . .
		Fault Insertion	Fault Insertion	. . .
		Decision Coverage	Decision Coverage	. . .
		Data Flow Coverage	Data Flow Coverage	. . .

System Test Level

SM-06	SG-07	SA-08	TV-10	Others . . .
Functional	Functional	Functional	Functional	. . .
Security	Security	Stress/ Volume	Stress/ Volume	. . .
		Boundary	Boundary	. . .
		Performance	Performance	. . .

Test Program Design

Test Program Design Models / Technique-Based Test Architecture

=> **associates** the **requirement** for test procedures with the test **techniques**.
=> provides the test team with a **clear picture of the test techniques** to be employed.
=> identifies t**he test requirements associated with each test technique.**
=> the test procedures that correspond to the applicable test **techniques/environments.**

Development Test Level

Error Handling	Path Coverage	Fault Insertion	Memory Leak	Others . . .
List Modules/ Units	List Modules/ Units	List Modules/ Units	List Modules/ Units	List Modules/ Units
. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .

System Test Level

Functional	Security	Stress/ Volume	Performance	Others . . .
Identify Test Procedure Series or Design Components	Identify Test Procedure Series or Design Components	Identify Test Procedure Series or Design Components	Identify Test Procedure Series or Design Components	Identify Test Procedure Series or Design Components
. . .	. . .	. . .	. . .	. . .

Test Program Design

Test Program Design Models / Effective Test Program Design

- The overall test program design involves both **dynamic and static test activities.**
- Effective use of test engineers' time to **support static test activities** => **improve** system application **design** and **development** => streamline the test effort => an effective test design => enables the test team to focus its efforts.
- The test architecture serves as a roadmap for the dynamic test effort.
- The dynamic test effort, in turn, facilitates the development and system-level test efforts.

=> These two test efforts are primarily based on **the use of white-box and black-box** approaches to testing.

Test Program Design

Test Program Design Models / Effective Test Program Design

White Box/Black Box Overview

White-Box Approach	Black-Box Approach
Test Approach	**Test Approach**
Structural testing approach focuses on application internals. Program-based.	Functional testing approach focuses on application externals. Requirements- or specification-based.
Characteristics	**Characteristics**
Module design	Functionality
Implementation	Requirements, use, standards
Do modules/functions meet functional and design specifications?	Correctness
Do program structures meet functional and design specifications?	Business forms, documents
How does the program work?	Does system meet business requirements?

Test Program Design

Test Program Design Models / Effective Test Program Design

White Box/Black Box Overview

Type of Test	Type of Test
Unit testing Integration testing	System testing User acceptance testing
Techniques	**Techniques**
• Fault Insertion • String Test • Error Handling • Statement Coverage • Decision Coverage • Condition Coverage • Path Coverage • Data Flow Coverage • Memory Leak • Cyclomatic Complexity	• Boundary Value Analysis • Cause-Effect Graphing • Random Testing • Error Guessing • Regression Testing • Stress (Volume) Testing • Replication Testing • Data Integrity Testing • Backup and Recoverability • Configuration Testing	• Performance Testing • Functional Testing • Security Testing • Equivalence Partitioning • Operational Readiness Testing • User Acceptance Testing • Compatibility/Conversion Testing • Benchmark Testing • Usability Testing • Alpha/Beta Testing
Personnel Involved	**Personnel Involved**
Developers and/or Test Engineers	Developers, Test Engineers, and End Users

Test Program Design

White-Box Techniques (Development-Level Tests)

White-box testing techniques are aimed at **exercising internal facets** of the target software program.

- **Not focus on identifying syntax errors.**
- Seek to **locate errors** that are more difficult to detect, find, and fix.
- Attempt to **identify logical errors and verify test coverage**.

=> Test procedures associated with the white-box approach make use of the control structure of the procedural design.

Test Program Design

White-Box Techniques (Development-Level Tests)

Test procedures provide several services:

1. They guarantee that **all independent paths** within a module have been **exercised at least once.**
2. They exercise **all logical decisions on their true and false sides.**
3. They execute **all loops at their boundaries and within their operational bounds.**
4. They exercise i**nternal data structures to ensure their validity.**

Test Program Design

White-Box Techniques (Development-Level Tests)

- White-box testing usually includes the **unit test strategy.**
- Can be called **unit testing, clear-box, or translucent testing** because of focusing on program's **internal** workings.
- This approach to testing is referred to as **a structural approach.**
- Examines the control **flow, all logical decisions, loops, upper and lower boundaries, and internal data structures.**

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Fault Insertion

- **Forcing return codes** to indicate errors and seeing how the code behaves.
- Simulate certain events, such as **disk full, out of memory, and so on**.
- Erroneous input testing: replacing ***alloc( )*** with a function that returns a NULL (10%) value => see how many crashes result.
- Checks the processing of both **valid and invalid inputs.**
- Values **outside the boundary range** of the parameters.

=> Fault insertion offers a way to measure the effectiveness of false tests.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Unit Testing

- Verifies that the **new code matches the detailed design.**
- Checks **paths** through the code.
- Verifies screens, pull-down menus, and messages formatted properly.
- Checks **inputs** for **range** and **type**.
- Verifies that each block of code **generates exceptions** or error returns.

=> Ensure that the **algorithms and logic are correct.**

=> Document logic, **overload** or **overflow** of range, **timing**, and **memory leakage** detection errors.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Error Handling Testing

- An error handler can **smooth the transition** with an **unexpected error.**
- Error handling tests seek to uncover instances where the system application **does not return a descriptive error message,** but instead **crashes** and reports a **runtime error.**
- Needs to be tested at the **development** test **level**.
- Some **invalid values** can be passed to the function to test the **performance** of error handling routines.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Memory Leak

- Focuses on the **execution** of the application.
- Attempt to find instances where the application is **not releasing or freeing up** allocated memory.
- A period of hours or days to see whether **memory consumption** continues to increase.
- Tools may also be able to identify the program statements where allocated memory is not released following use of the memory space.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Integration Testing

- Verify that each software **unit interfaces correctly** to other software units.
- May utilize a **top-down or bottom-up** structured technique.
- Examine the parameters passed between two components and the global parameters.
- Some integration testing may take place during unit testing.
- Classified in a range of 1 to 4, based on their degree of severity (1 being the most critical and 4 being the least).

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

String Testing

- Involves the **examination** of **a related group of modules** that constitute a software function.
- Ensures **sufficient testing** of a system's components.
- Determines whether modules **work successfully** to form a **cohesive unit.**
- Whether the software unit produces **accurate** and **consistent results**.
- String testing concentrates on the **interactions** of the **functions**.
- Verifies that each program **statement** executes **at least once.**
- Executes **all loops and conditional statements**, **conditions**, **boundary conditions**.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Coverage Analysis

- ***Statement Coverage***
  - Often referred to as C1 (node coverage).
  - Reports whether each **executable statement** is **encountered**.
  - Verifies **coverage** at a **high level**, rather than decision execution or Boolean expressions.
  - Requires that **every statement** in the program be **invoked at least once.**
  - A **weakness** of this technique is that it does **not verify decisions** (paths/results), **not test** Boolean **expressions thoroughly**.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Decision Coverage

- Seeks to identify the percentage of all possible decision outcomes exercised
- Referred to as branch coverage and is denoted as C2.
- Requires that **every point of entry and exit** in the software program be **invoked at least once.**
- Requires **all possible conditions for a decision** in the program be **exercised at least once.**
- Requires that **each decision** in the program be tested using all possible **outcomes at least once.**

=> **Weakness**: **inadequate** for high-integrity applications, **incorrect** evaluation of a condition.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Condition Coverage

- Similar to decision coverage.
- Seeks to verify the **accuracy** of the true or false outcome of each Boolean subexpression.
- Employs tests that measure the **subexpressions independently of each other.**
- The results of these measures are similar to those obtained with decision coverage except that the former show greater sensitivity to the control flow.

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Path Coverage

- Seeks to verify whether each of the **possible paths** in each function has **executed properly**.
- A path is a set of branches of logic flow.
- Boundary-interior path testing considers **two possibilities** for loops: **zero repetitions and more than zero repetitions.**
- Provides very thorough testing, but has two **significant drawbacks**:
  - the **number** of **possible paths** may be **exceedingly exhaustive** and beyond **the scope of most test programs.**
  - **very time-consuming.**

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Data Flow Coverage

- Is a variation of path coverage.
- Seeks to **incorporate** the flow of data into the selection of test procedures.
- Based on the selection of test path segments that **satisfy** some **characteristic of the data flow for all possible types of data objects.**

Test Program Design

White-Box Techniques / Descriptions of White-Box Techniques

Branch Coverage

- Branch coverage measures **the number of times that logical branches have been exercised for both true and false conditions.**
- This analysis is used most **often** for **detailed unit testing of systems.**

Test Program Design

White-Box Techniques / Automated Tools Supporting White-Box Testing

White-Box Test Techniques and Corresponding Automated Test Tools

White-Box Test Techniques	Automated Test Tools
• Fault Insertion	• Coding Tools
• String Testing	• GUI and Server Test Tools
• Error Handling	• Error Handling Tools
• Statement Coverage	• Static and Dynamic Analyzers
• Decision Coverage	• Static and Dynamic Analyzers, Coverage Analysis Tools
• Condition Coverage	• Static and Dynamic Analyzers, Coverage Analysis Tools
• Path Coverage	• Static and Dynamic Analyzers, Coverage Analysis Tools
• Test Data Flow Coverage	• Data Modeling Tools, Flow Diagram Editors
• Memory Usage	• Memory Usage Tools
• General Resource Usage	• General Resource Usage Tools
• Code Complexity	• Static and Dynamic Analyzers, Source Code Analyzers
• Cyclomatic Complexity	• Cyclomatic Complexity Analyzers, other Software Metrics Tools

Test Program Design

Black-Box Techniques (System-Level Tests)

Black-box testing is testing only via **established, public interfaces** such as the user interface or the published application programming interface (API).

- White-box testing - the program's internal workings <=> Black-box testing - the application's behavior against requirements.
- Seek to investigate **three basic types of errors:**
  - Errors associated with the functional paths supported by the software.
  - The **computations** performed by the software.
  - The **range or domain of possible data** values that can be executed by the software.

Test Program Design

Black-Box Techniques (System-Level Tests)

- The testers do **not** primarily **concern** themselves with the **inner workings** of the software components, though the inner components of the software are nevertheless exercised by default.
- **Concerned** about the **inputs and outputs of the software.**
- Black-box testing is considered to be synonymous with system testing.
- Can also **occur** during **unit or integration testing.**
- **User participation** is **important** to black-box testing because of their expectation from the business functions.
- The **correctness** of **data** is **key** in successfully completing the system testing.

Test Program Design

Black-Box Techniques (System-Level Tests)

Black-box testing attempts to **find errors in many categories**, including:

- Incorrect or missing functionality.
- Interface errors.
- Usability problems.
- Errors in data structures or external database access.
- Performance degradation problems and other performance errors.
- Loading errors.
- Multi-user access errors.
- Initialization and termination errors.
- Backup and recoverability problems.
- Security problems.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Equivalence Partitioning

- Three basic types of equivalence classes apply when testing for range and domain errors: *in-bound, out-of-bound, and on-bound* situations.
- Examine boundary cases plus/minus one to avoid missing the "*one too many*" or "*one too few*" error.
- The test team should perform exploratory testing.
- Test procedures that should result in an error when executed are referred to as negative cases.
- **Advantage**: the scope of exhaustive testing is reduced.
- **Disadvantage**: the resulting test procedures do not include other types of tests that have a high probability of finding an error.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Boundary Value Analysis

- Can be applied to both **structural** and **functional** testing levels.
- Boundaries define **three classes of data: good, bad and on the border.**
- Endpoints, maximum/minimum values.
- Outside the boundary of values - that is, invalid values.
- For example: numeric and alphabetic values.

When the number of possibilities is very large and possible results are very close in value, choose input values that give largest variances in output - that is, *perform sensitivity analysis.*

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Cause/Effect Graphing

- Provides a concise representation of logical conditions and corresponding actions.
- Four steps:
  - (1) The listing of causes (input conditions) and effects (actions).
  - (2) A cause-effect graph is developed.
  - (3) The graph is converted to a decision table.
  - (4) the identification of causes and effects by reading the functional specifications.
- The causes are listed vertically on the left-hand side and link with the effects are listed on the right-hand side.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

System Testing

- Used as a synonym for “black-box testing because of the application’s “externals” concerns.
- Encompasses several testing subtypes:
  - Functional.
  - Regression.
  - Security, stress, performance, usability, random, data integrity, conversion, backup and recoverability.
  - Configuration, operational readiness, user acceptance, and alpha/beta testing.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Functional Testing

- With the intent of discovering nonconformance with end user requirements.
- Is **central** to most software test programs to check if application does in accordance with specified requirements.
- When working independently and sharing the same data or database, make sure that test engineer A does **not modify** or **affect** the data being manipulated by test engineer B, **potentially invalidating** the test results produced by test engineer B.

=> Automated test procedures should be organized in such a way that effort is not duplicated.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Functional Testing

- The test team should review the test plan and the test design:
  - Determine the **order** or **sequence** in which specific transactions must be tested, either to accommodate database issues or as a result of control or workflow.
  - Identify any **patterns** of **similar actions** or events that are used by multiple transactions.
  - Review **critical and high-risk functions** so as to place a greater priority on tests of this functionality and to address associated test procedures early in the development schedule.
  - Create a modularity-relationship matrix (Chapter 8).

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Regression Testing

- Aimed at **finding** and **documenting defects and tracking them to closure.**
- Make sure f**ixes/changes not to create new errors/issues.**
- Run with **no manual intervention.**
- Once the application is somewhat stable, the test team should focus on automating some or all regression tests, especially those involving high-risk functionality, repetitive tasks, and reusable modules.
- Regression tests can also be **reused many times during the development life cycle** for stress, volume, and performance testing.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Security Testing

- Verify the **proper** performance of system **access** and **data access** mechanisms.
- Subvert the program's security checks.
- To validate security levels and **access limits** and thereby verify compliance with specified security requirements and any applicable security regulations.
- Security testing might involve tests of a **COTS** product that is integrated into the application to support security.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Stress Testing

- With the intent to **discover** the as-built **limitations of the system**.
- Performed when processing of transactions reaches **its peak** and **steady loads** of **high-volume data** are encountered.
- Measures the capacity, resiliency of the system on each hardware platform.
- Multiple users exercise specific functions concurrently and some use values **outside of the norm.**
- Process a **huge amount of data** or perform **many function calls within a short period of time.**

=> Use tools to create virtual users => capture response time => track any performance degradation.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Stress Testing

- High-stress scenarios:
  - Operations involving complex queries.
  - Large query responses.
  - Large data object retrievals.
  - The concurrent operation of a large number of test procedures.
  - Long hours of running.
- Monitor resource usage:
  - Global memory, DOS memory, free file handles, and disk space.
  - Identify trends in resource usage so as to detect problem areas: memory leaks and excess consumption.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Stress Testing

- **Unit stress** tests generate stress on a single interface.
- **Module stress** testing verifies that business functions are processed in a common area.
- **System stress** testing involves loading the system using high-volume transactions.
- Include data volume tests, concurrency tests.
- Some scalability tests need to be conducted.
- Database time, response time, and central processing unit (CPU) time.
- Minimum transaction response time, maximum transaction response time, mean transaction response time, number of parsed transactions, and number of failed transactions.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Stress Testing

- The test team needs to **develop** a **benchmark** plan to support stress testing.
- The **benchmark** should includes: **month, day, and week values based upon functional requirements.**
- The test team also needs to determine occurrence **rates** and **probabilities**.

=> Benefits of stress testing include the identification of **threshold** failures, such as **limitations** involving not enough memory, SWAP or TEMP space, maximum number of concurrent open files, maximum number of concurrent users, maximum concurrent data, and DBMS page locking during updating.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Performance Testing

- Verify that the system application **meets specific performance** efficiency objectives.
- Performance testing can measure and report on such data as input/output (I/O) rates, total number of I/O actions, average database query response time, and CPU utilization rates.
- **The same tools used in stress testing** can generally be used in performance testing to allow for automatic checks of performance efficiency.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Performance Testing

- To conduct performance testing, the following performance objectives need to be defined:
  - How many transactions per second need to be processed?
  - How is a transaction defined?
  - How many concurrent users and total users are possible?
  - Which protocols are supported?
  - With which external data sources or systems does the application interact?

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Usability Testing

- Verify that the system is easy to use and that **the user interface appearance is appealing.**
- Such tests consider the **human element** in system operation.
- the test engineer needs to evaluate the application from **the perspective of the end user.**

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Usability Test Considerations

General On-Line/Screen Testing

Mode and commands
Confirmation required for deletions
Layout conforms to standards
Screen field names conform to standards
Positive/negative/null responses to entries
Menu flows
Backout of possible end-user errors
Lockout keys that could alert the end user
Spelling grammar, accepted terms
Same name for same fields
Data obscured by system messages
Logical flow of screens
Standard hotkey usage
Ability to save user's work

Error Messages

User-oriented, easy to understand
Appropriate messages
Standardized messages
Account for known errors

Edits—Numeric

Valid values
Maximum/minimum or table
Leading zeroes, decimals
Proper sign
Field conforms to requirements
Defaults
Integer versus floating

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Usability Test Considerations

Memory allocations
Meaningful system messages
Type ahead runaway
GUI considerations
- Icons (de)selection
- Graphic reflects data
- Backout/cancel options
- Horizontal/sidebar functions

Colors

Cursor

At first field requiring data
Skip right fields during edits
Behavior in locked on noneditable fields
Movement from one field to the next
Edit order of fields

Edits—Dates

Format
Display back after fill-in
Range checks, leap year, year 2000, and so on
Blank, spaces, nulls
Field conforms to requirements
Defaults

Edits—Character

Format
Range, domain
Display back after fill-in
Field conforms to requirements
Defaults
Leading/trailing blanks

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Usability Test Considerations

Lists

Garbage/blanks
Too many/too few entries
Omitted/invalid delimiters
Too much /too little visible

Reports

Sort order
Display order
Date printed/page numbers
Data truncation versus wrapping in columns
Consistency in reports

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Random Testing

- Consist of spontaneous tests identified by the test engineer during test development or execution.
- Referred as *monkey* tests.
- No formal design, nor are the tests commonly **rerun as part of regression testing.**
- **Documented => developers understand** what test **sequence** was **executed** during random testing when a defect was discovered.
- Insufficient for validating complex, safety-critical, or mission-critical software.
- Best applied as a complementary test strategy.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Data Integrity Testing

- Verifies that data are being stored by the system in a manner where the data is **not compromised by updating, restoration, or retrieval processing.**
- Checking data fields for alphabetic and numeric characters, for information that is too long, and for correct date format.
- Data consistency checks include both **internal** (correct data type) and **external** (relational integrity) validations of essential data fields.
- To uncover design flaws that may result in **data corruption, unauthorized data access, lack of data integrity across multiple tables, and lack of adequate transaction performance.**

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Conversion Testing

- Measures and reports the **capability** of the software to **convert existing application data to new formats.**
- Conversion accuracy is measured by comparing the test data file dump with the new database.
- Conversion testing will be **done separately as part of the initial database load software test.**

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Backup and Recoverability Testing

- Verifies that the system **meets specified backup and recoverability requirements.**
- To prove that the database and software can **recover** from partial or full catastrophic **failures** of system hardware or software.
- To determine the **durability** and **recoverability** levels of the software on each hardware platform.
- Recovery testing is to discover the extent to which data can be **recovered** after a system **breakdown**.
- **Recover** all of the data or just part of it?
- **How** much can be recovered and how?
- Are the recovered data still correct and **consistent**?

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Configuration Testing

- Verifies that an application **operates properly** on machines with **different hardware and software configurations.**
- Check for **compatibility issues** and help to determine the optimal configuration of hardware and software to support an application.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Operational Readiness Testing

- Helps determine whether a system is ready for n**ormal production operations.**
- **All valid and invalid values are defined and applied** during these tests.
- It is very important to test all possible invalid values to verify that the AUT will meet the project's specifications.
- Installed on its **targeted hardware platforms** using the documentation provided by the development/test group.
- **Uninstall** instructions are tested for their impact on the **environment**.
- Check if components are **integrated** and **communicating** properly.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

User Acceptance Testing

- Getting the user involved early in the testing process pays off at this stage.
- Includes testing performed for or by **end users** of the software product.
- Ensure that end users are **satisfied** with the functionality and performance.
- **In a controlled environment:** where end users are asked to make a determination of whether to accept or reject.
- **In an uncontrolled environment:** to solicit end-user feedback.

The acceptance test phase begins only after the successful conclusion of system testing and the successful setup of a hardware and software configuration to support acceptance testing, when this configuration differs from the system test environment.

Test Program Design

Black-Box Techniques (System-Level Tests) / Black-Box Technique Descriptions

Alpha/Beta Testing

- Most software vendors use alpha and beta testing to **uncover errors through testing by the end user.**
- Customers at the developer's site, with the developer present, usually conduct alpha testing.
- Beta testing is usually conducted at one or more customer sites with the developers not present.

Test Program Design

Black-Box Techniques (System-Level Tests) / Automated Tools Supporting Black-Box Testing

Black-Box Test Techniques	Automated Test Tools
• Equivalence Partitioning	• Develop program code to perform tests
• Boundary Value Analysis	• Develop program code to perform tests
• Cause-effect Graphing	• Flow Graphing Tools
• Random Testing	• GUI Test Tools
• Error Guessing	• GUI Test Tools
• Regression Testing	• GUI/Server Test Tools
• Stress Testing	• Load Test Tools
• Replication Testing	• Load Test Tools
• Data Integrity Testing	• Data Analysis Tools
• Backup and Recoverability Testing	• Load Test Tools/GUI Test Tools/Server Test Tools
• Configuration Testing	• Multiplatform Test Tools
• Performance Testing	• Load Test Tools
• Functional Testing	• Load Test Tools/GUI Test Tools/Server Test Tools
• Security Testing	• Security Test Tools
• Operational Readiness Testing	• Load Test Tools/GUI Test Tools/Server Test Tools
• User Acceptance Testing	• GUI Test Tools
• Compatibility/Conversion Testing	• Load Test Tools/GUI Test Tools/Server Test Tools
• Benchmark Testing	• Benchmarking Tools
• Usability Testing	• Usability Measurement Tools
• Alpha/Beta Testing	• Load Test Tools/GUI Test Tools/Server Test Tools

Test Program Design

Test Design Documentation

Document	Function
Verification method as part of the requirements traceability matrix (RTM)	The proper implementation of system and software requirements can be verified in several ways. When the verification method for a requirement consists of a "test," then the associated test requirement needs to be factored into test design. Verification methods may be identified in a requirements traceability matrix developed using a spreadsheet or in an automated fashion using a requirements management (RM) tool.
Test requirements	Test requirements maintained in a document or in an automated fashion using a RM tool are essential in supporting test design.
Test procedure template	A test procedure template is used as a baseline when designing a test procedure format for a particular project.
Software application critical path diagram	This information aids in the prioritization of test procedures.

Test Program Design

Test Design Documentation

Risk analysis documentation	This information aids in the prioritization of test procedures.
Decision logic table	This table helps to understand conditional and logic flow within software modules.
Detailed test schedule	This schedule provides information on the kinds of tests that need to be performed, the planned duration of such tests, and the duration of test phases within the test life cycle. This information aids the prioritization of test procedure development.
Manual versus automated test considerations	The test team should perform and document analysis to determine the tests that will be automated and those that will be performed manually.
Test procedure design standards	The test team should refer to standard guidance on test procedure design.
Input data file requirements	The test team should perform and document analysis to determine the necessary white-box and black-box data sources. The outcome of this analysis is beneficial when designing test procedures.

Test Program Design

Test Design Documentation

Data flow diagram/ data dictionary	This information aids in the identification of test data sources.
Use case analysis	Use case scenario information is beneficial when designing test procedures. Some CASE/RM tools have mechanisms to automatically generate test procedures.
Environment	The test team should document dependencies between test procedures and the applicable technical environment. It needs to identify any differences in the technical environment for each test life cycle phase (unit, integration, system, acceptance).
Outputs	Reports, graphs, data, and test logs may serve to document test runs. They are helpful in redesigning test procedures.
Documentation of build (version) of the software	While designing test procedures, it is important to have documentation of the version of the software, including information about the functionality that the version of software contains, the fixes included in the new version of software, and so on.
Benchmarks with review and approvals	Benchmarks are the expected level of software quality that has been reviewed by management and approved. They lay the foundation for expectations of test procedures and quality of the software.
Test procedure modularity-relationship matrix	See Section 8.1 for details.

Test Procedure Design

Test Procedure Design

After test requirements have been derived, test procedure design can begin

Test procedure definition consists of:

- The **definition** of **logical groups** of test procedures.
  - Each test procedure is then **identified** as either an **automated** or a **manual** test.
  - Number of test **techniques** to be employed.
  - The t**otal number of test procedures** is required.
  - **Estimates** of the number of test procedures to be performed.
- **A naming convention** for the suite of test procedures.

Test Procedure Design

Step	Description
1	**Test architecture review.** The test team reviews the test architecture to identify the relevant test techniques.
2	**Test procedure definition (development level).** A test procedure definition is constructed at the development test level that identifies the test procedure series relevant to the different design components and test techniques.
3	**Test procedure definition (system level).** A test procedure definition is constructed at the system test level that identifies the test procedure series relevant to the different test techniques.
4	**Test procedure design standards.** Design standards are adopted, and a naming convention is created that uniquely distinguishes the test procedures on the project from test procedures developed in the past or on other projects.
5	**Manual versus automated tests.** Test procedures will be depicted as being performed either manually or as part of an automated test.
6	**Test procedures flagged for detailed design.** Test procedures that stand out as more sophisticated are flagged. These test procedures are further defined as part of detailed test design.
7	**Detailed design.** Those test procedures flagged in step 6 are designed in further detail within a detailed test design file or document. This detailed design may consist of pseudocode of algorithms, preliminary test step definition, or pseudocode of test automation programs.
8	**Test data mapping.** The test procedure matrix is modified to reflect test data requirements for each test procedure.

Test Procedure Design

Test Procedure Definition

Test procedures address:

- **Preconditions** for a test.
- **Data** inputs necessary for the test.
- **Actions** to be taken.
- Expected **results**.
- **Verification methods.**

Test is to **find errors** => tests that have a **high probability** of finding previously **undiscovered errors** => cover **expected inputs** and **outputs**, account for unexpected input and output values.

Test Procedure Design

Test Procedure Definition / Sample Test Architecture

Development Test Level

SM-06	SG-07	DC-08	SA-09	TV-10
Error Handling	Error Handling	Error Handling	Error Handling	Error Handling
Memory Leak	Memory Leak	Memory Leak	Memory Leak	Memory Leak
		Path Coverage	Path Coverage	Path Coverage
		Fault Insertion	Fault Insertion	Fault Insertion
		Decision Coverage	Decision Coverage	Decision Coverage

Test Procedure Design

Test Procedure Definition / Sample Test Architecture

System Test Level

Functional	Security	Stress/ Volume	Performance	Usability
SM-06 SG-07 DC-08 SA-09 TV-10	SM-06 SG-07 and Security Plan Requirements	TV-10	TV-10	SM-06 SG-07 DC-08 SA-09 TV-10

The **test architecture** provides the test team with **a clear picture** of the test techniques that need to be employed.

Test Procedure Design

Test Procedure Definition / Test Procedure Definition (Development Test Level)

TP Numbering Allocation	Design Component ID	Test Technique	Number of Test Procedures
100–150	SM601–SM634	Error Handling	35
151–199		Memory Leak	35
200–250	SG701–SG728	Error Handling	30
251–299		Memory Leak	30
300–350	DC801–DC848	Error Handling	50
351–399		Memory Leak	50
400–599		Path Coverage	200
600–650		Fault Insertion	50
651–849		Decision Coverage	200

Test Procedure Design

Test Procedure Definition / Test Procedure Definition (Development Test Level)

850–899	SA901–SA932	Error Handling	35
900–950		Memory Leak	35
951–1150		Path Coverage	200
1151–1199		Fault Insertion	35
1200–1399		Decision Coverage	200
1400–1450	TV1001–TV1044	Error Handling	45
1451–1499		Memory Leak	45
1500–1699		Path Coverage	200
1700–1750		Fault Insertion	45
1751–1949		Decision Coverage	200
1950–1999		Integration Test	25
			Total = 1,745

Test Procedure Design

Test Procedure Definition / Test Procedure Definition (System Test Level)

TP Numbering Allocation	Test Technique	Number of Units or Threads	Number of System Require-ments	Number of Test Require-ments	Number of Test Procedures
2000–2399	Functional	186	220	360	360
2400–2499	Security	62	70	74	74
2500–2599	Stress	4	12	24	96
2600–2699	Performance	4	14	14	56
—	Usability	186	4	4	—
					586

Usability tests will be conducted as part of functional testing, and, as a result, no additional test procedures are developed for this test technique.

Test Procedure Design

Test Procedure Definition / Test Procedure Definition (System Test Level)

- For **functional and security tests**, there may be one test procedure for **every test requirement.**
- For **stress and performance testing**, four threads will be altered for each test procedure so as to examine 12 or 14 different system requirements or use cases.
- By using the two levels and capturing two different measurements, the test team would be able to **examine the performance degradation** between the two levels.

Test Procedure Design

Test Procedure Definition / Test Procedure Naming Convention

Naming Convention	Design Component/ Test Technique	Test Level	Test Procedure Estimate
WF100–WF199	System Management (SM)	Development	70
WF200–WF299	Security Guard (SG)	Development	60
WF300–WF849	Distributed Computing (DC)	Development	550
WF850–WF1399	Support Applications (SA)	Development	505
WF1400–WF1949	Active Trade Visibility (TV)	Development	535
WF1950–WF1999	Integration Test	Development	25
WF2000–WF2399	Functional/Usability Tests	System	360
WF2400–WF2499	Security	System	74
WF2500–WF2599	Stress	System	96
WF2600–WF2699	Performance	System	56
WF2700	System Test Shell	System	1

Test Procedure Design

Test Procedure Definition / Automated Versus Manual Test Analysis

- Tests at the white-box comprise primarily automated tests.
- Tests at the system level represent a combination of automated and manual tests => categorize automated tests and manual tests.

=> If it takes **too much effort and time** => manual tests might be better.

=> Test automation **goals** is to avoid **duplicating the development effort.**

Test Procedure Design

Test Procedure Definition / Step by Step - Don’t Try to Automate Everything at Once

- **No experience** => more **cautious** approach for introducing automation.
- It is better to apply an **incremental approach** toward increasing the depth and breadth of test automation.

Test Procedure Definition / Step by Step - Not Everything Can Be Tested or Automated

- It is **not feasible** to automate every required test given schedule and budget constraints.
- E.g: Impossible to automate the verification of a print output => what if simply out of paper?

Test Procedure Design

Test Procedure Definition / Don’t Lose Sight of Testing Goals

- The overall testing **goal**: **to find defects early.**
- The automation effort might **postpone** the **immediate discovery** of the **defects** because the test engineer is **too involved** in creating **complex** automated test scripts.

Test Procedure Definition / Don’t Duplicate Automation of the AUT’s Program Logic

- If it takes **more effort to automate** a test script as it did to code the function, a new testing approach is required.
- A problem with AUT’s program logic => automated scripts cannot determine the error logic of AUT’s program.

Test Procedure Design

Test Procedure Definition / Analyze the Automation Effort

- Automation effort should be based on the **highest-risk functionality.**
- If **only two weeks are allotted** for test development and execution => not to use an automated test tool at all.

Test Procedure Definition / Analyze the Reuse Potential of Automated Modules

- Keep **reuse** in mind. If not => automation efforts are wasted.
- Automation may still permit test execution and regression test schedule savings in the time and budget considerations.

Test Procedure Design

Test Procedure Definition / Focus Automation on Repetitive Tasks - Reduce the Manual Test Effort

- If **repetitive** tasks are automated, test engineers can be **freed** up to test more **complex functionality**.
- E.g: “the system should allow for adding 1 million account numbers.” => must implement automated tests.

Test Procedure Definition / Focus Automation on Data-Driven Tasks - Reduce the Manual Test Effort

- Performing **repetitive** test tasks manually is the fact that such manual efforts are **prone to errors**.
- Computers and software programs do repetitive tasks better than human.

Test Procedure Design

Test Procedure Definition / Consider the Test Tool's Capabilities

- The test engineer needs to keep in mind the **test tool's capabilities.**
- Client-side GUI tests is **viewed as different** from server-side tests.

Test Procedure Definition /Automate Test Requirements Based on Risk

- Automation **relies on risk analysis** (highest-risk functionality).
- The **schedule sequence** is generally **based on risk, among other issues.**

Test Procedure Design

Test Procedure Definition / Automated Versus Manual Tests

TP Number	Design Component	Test Technique	SR ID	SWR ID	TR ID	Verification Method
2330	TV1016	Functional	3.2.3c	TV029	2220	A
2331	TV1016	Functional	3.2.3c	TV030	2221	A
2332	TV1016	Functional	3.2.3c	TV031	2412	M
2333	TV1017	Functional	3.2.3d	TV032	2222	A
2334	TV1017	Functional	3.2.3d	TV033	2412	A
2335	TV1018	Functional	3.2.3e	TV034	2223	A
2336	TV1018	Functional	3.2.3e	TV035	2412	M
2337	TV1019	Functional	3.2.3f	TV036	2224	A
2338	TV1019	Functional	3.2.3g	TV037	2412	A
2339	TV1019	Functional	3.2.3g	TV038	2225	A
.	.	.	.	.	.	.

Test Procedure Design

Automated Test Design Standards / When to Design

- Performed in **parallel** with the application development effort.
- Test requirements and test design can be initially addressed during the application **requirements-gathering phase.**

=> Whether an application design is **testable** and influence the **testability** of the resulting application code.

=> The test design effort **should not interrupt** application development work, but should be **integrated smoothly** into the application development life cycle.

Test Procedure Design

Automated Test Design Standards / What to Design

Good test procedures should not only account for what the system should do, but also include test exercises to verify **performance** for **unexpected** conditions.

- Has an automated versus manual test analysis been conducted and documented?
- What is the sequence of actions necessary to satisfy the test requirements?
- What are the inputs and expected outputs for each test procedure?
- What data are required for each test procedure used?
- How is the function's validity verified?
- What classes of input will make for good test procedures?
- Is the system particularly sensitive to certain input values?
- How are the boundaries of a data class isolated?
- What data rates and data volume can the system tolerate?
- What effect will specific combinations of data have on system operation?

Test Procedure Design

Automated Test Design Standards / How to Design

1. Cover the **important aspects** of the application.
2. Comply with **design standards** (templates and naming conventions).
3. Employ the test design techniques discussed so far to **derive test requirements.**
4. Use these test requirements as a **baseline** for test procedure design.
5. Make the automated test **reusable** and **repeatable** and **helpful** to detect **errors** or **defects** in the target software.
6. Tackle the **high-priority tasks first.**
7. Solicit **customer and end-user involvement** early in test design => avoid surprises during test implementation.

Test Procedure Design

Automated Test Design Standards / Test Procedure Modularity

- Test procedure design standards or guidelines need to **address the size** of a test.
- It is beneficial to **limit the scope of a test procedure to a single function**, so it will remain **manageable** and **maintainable**.

Test Procedure Design

Automated Test Design Standards / Test Procedure Independence

- It is a good idea to **avoid data dependency** between test procedures, whenever feasible => avoid domino effects.
- Whenever a data dependency exists, make sure to document it in the **modularity-relationship matrix** (Chapter 8).
- It is also important to avoid creating test procedures in a **context-dependent fashion** (one ends, the other begins).
- Make sure to document any test procedure context dependency in the modularity-relationship matrix.

Test Procedure Design

Automated Test Design Standards / Scripting Language

- The test organization will need to **document** within its test design standards which scripting language has been adopted.

=> This choice **eliminates any dependencies** and allows any member of the team to read and interpret the test procedure code that is generated.

Test Procedure Design

Automated Test Design Standards / Test Tool Database

- The test team must evaluate the **benefits** and **drawbacks** to each particular database.
- Should consider the **rate of database corruption problems** encountered, any database record size limitations, requirements for ODBC connections, and the use of SQL statements.

Test Procedure Design

Automated Test Design Standards / Test Procedure Template

Template Example (Automated Tests)

Test Procedure ID	ACC002
Name of Module	Monthly statement generation
Name of Function	Calculate the monthly payment
Name of Programmer	Michael Montgomery
Functional Cross-Reference	2.2.1. Create monthly statements Detail: Input and output from each function is tested by verifying the interactions with other functions in the module. Each function is tested for integration and all the assumptions are verified.

Test Procedure Design

Automated Test Design Standards / Test Procedure Template

Function	Description	Expected Result	Date Executed	Actual Result
ID of environment setup script to be run	Env001 setup script needs to be run to play back this test procedure	Environment is in the expected state		
Starting location/ end location	Main application window			
ID or name of test procedure that must be executed prior to the start of this test procedure	ACC001 adds the new account 001	ACC001 run successfully		
Scope	Calculates interest rate of account 001			

Test Procedure Design

Automated Test Design Standards / Test Procedure Template

Function	Description	Expected Result	Date Executed	Actual Result
Input	Inputs an amount and compares it to make sure the amount entered is greater than 0	Amount > 0	10/10/98	Amount > 0; pass
Action taken	Checks whether the valid interest rate is calculated and retrieved	Interest rate = expected result	10/10/98	Fails intermit-tently
Test requirement	ACC143			
External data file (or the data or state of the data that should already exist in the database being used for testing)	Test procedure ACC001 will put data in the correct state			
Name of specification file	Functional specs B			

Test Procedure Design

Automated Test Design Standards / Naming Conventions

- It is very important to follow a naming convention to
  - **Prevent duplication** of test procedure IDs.
  - **Avoid duplications** of test procedures.
  - **Quickly ascertain** the kind of test being performed.
  - Make the entire test suite more **maintainable**.
  - Make procedures **manageable** and **readable**.

Test Procedure Design

Automated Test Design Standards / Naming Conventions

Module to Be Tested	Letter Denoting the Module
Employee personal information	E
Employee leave information	L
Employee pay information	P
Employee benefit information	B

First letter in the file naming convention

Test Procedure Design

Automated Test Design Standards / Naming Conventions

Source of Test Script	Letter Denoting Source
Functional requirement	**F**
Detail requirement	**D**
System integration	**S**
Design verification	**V**

Second letter in the file naming convention

Test Procedure Design

Automated Test Design Standards / Naming Conventions

Functionality	Letters Denoting Functionality
Add	AD
Edit	ED
Security	SE
Menu	MN
Help	HE
Delete	DE
Graphs	GR
Report	RE

Third and fourth letters in the file naming convention

Test Procedure Design

Automated Test Design Standards / Naming Conventions

Name of the Test Script Developer	Letters Denoting Test Engineer
Debbie Trekker	DT
Jack Burner	JB
Rick Black	RB

Fifth and sixth letters in the file naming convention

Test Procedure Design

Automated Test Design Standards / Naming Conventions

Name of Test Procedure	Interpretation
EFADDT07	Employee personal information, to meet a functional requirement, for add capability, developed by Debbie Trekker and seventh in the sequence.
PVSERB13	Employee pay information, verification of development, security module, developed by Rick Black and thirteenth in the sequence

Test Procedure Design

Manual Test Design Guidelines

- **Expected Output**
  - Will tests require screen prints?
  - Will tests require sign-off by a second test engineer who observes the execution of the test?

Test Procedure Design

Manual Test Design Guidelines

- **Naming Convention**
- **Test Procedure Detail**
  - Test Procedure ID
  - Test Procedure Name
  - Test Author
  - Verification Method
  - Action
  - Criteria/Prerequisites
  - Dependency
  - Requirement Number
  - Expected Results
  - Actual Results
  - Status

Test Procedure Design

Manual Test Design Guidelines

- Manual Test Procedure Example
  The details of the test procedure are either **generated by the system or completed by a test engineer.**
  - Object Level (system-generated) - Shows the hierarchy and relationships of test procedures.
  - Object Number (system-generated)
  - Object Identifier (system-generated) - Links defects to each test procedure.
  - Absolute Number (system-generated)
  - Created by (system-generated)—Gives the name of the test engineer generating the test procedure.

Test Procedure Design

Manual Test Design Guidelines

- Manual Test Procedure Example
  The details of the test procedure are either **generated by the system or completed by a test engineer.**
  - Created on (system-generated) - gives a date.
  - Created Thru (system-generated)
  - Criteria/Prerequisites - Includes criteria or prerequisite information completed by the test engineer before the test procedure could be run (such as specific data setup necessary).
  - Expected Results - Explains the expected results of executing the test procedure.

Test Procedure Design

Manual Test Design Guidelines

- Manual Test Procedure Example
  The details of the test procedure are either **generated by the system or completed by a test engineer.**
  - Actual Result - Lists “Same as Expected Result” as the default, but is changed if the test procedure fails.
  - Last Modified by (system-generated)
  - Last Modified on (system-generated)
  - Object Heading (system-generated)
  - Object Short Text
  - Object Text

Test Procedure Design

Manual Test Design Guidelines

- Manual Test Procedure Example
  The details of the test procedure are either **generated by the system or completed by a test engineer.**
  - Precondition/Dependency - Filled in if the test procedure has a dependency on another test procedure or when two test procedures would conflict with each other if run at the same time.
  - Requirement Number - Gives the number of the system or software requirement that applies.
  - Status - Identifies the status of the executed test.
  - Step - Documents the steps needed to create a test procedure. It is equivalent of pseudocode in software development.

Test Procedure Design

Manual Test Design Guidelines

- Manual Test Procedure Example
  The details of the test procedure are either **generated by the system or completed by a test engineer.**
  - Test Procedure ID - Defines the naming convention used to document the test procedure ID.
  - Test Procedure Name - Provides a full description of the test procedure.
  - Verification Method - Automated or manual test, inspection, analysis, demonstration, or certification.

Test Procedure Design

Detailed Test Design

The test team could begin printing out a list of all planned system-level test procedures => a clear picture of how many test procedures.

TP Number	Design Component	Test Technique	SR ID	TR ID	A/M	Detailed Design
2330	TV1016	Functional	3.2.3c	2220	A	—
2331	TV1016	Functional	3.2.3c	2221	A	—
2332	TV1016	Functional	3.2.3c	2412	M	—
2333	TV1017	Functional	3.2.3d	2222	A	DD
2334	TV1017	Functional	3.2.3d	2412	A	—
2335	TV1018	Functional	3.2.3e	2223	A	DD
2336	TV1018	Functional	3.2.3e	2412	M	—
2337	TV1019	Functional	3.2.3f	2224	A	—
2338	TV1019	Functional	3.2.3g	2412	A	DD
2339	TV1019	Functional	3.2.3g	2225	A	—

Test Procedure Design

Detailed Test Design

Test procedures should be more consistent and include all of the tests required.

Pseudocode/sequence of steps may be included in test design.

Section	Description
1. Introduction	Describes the purpose of the document and the parameters by which the particular test procedures were selected to be included as part of the detailed design effort.
2. Test Procedure List	Lists test procedures for which the detailed design applies. The rationale for requiring a detailed design of the test procedure should be provided.
3. Detailed Design	Includes a detailed design for each test procedure. The detailed designs should be provided in order by test procedure (alphanumeric) number.
4. Summary	Summarizes key points and notes pertaining to the design and subsequent development of the test procedures.

Test Procedure Design

Test Data Requirements

- Once test data requirements are outlined, the test team should plan the **means for obtaining, generating, or developing the test data.**
- The project test plan needs to identify the **names and locations of the applicable test databases and repositories** necessary to exercise software applications.
- The mechanism for refreshing the test database to an original **baseline** state - a necessity in **regression** testing - also needs to be **documented** within the project test plan.

Test Procedure Design

Test Data Requirements / White-Box Test Data Definition

The test data should be for:

- **Executing** every program statement **at least once.**
- **Assuring** that each **condition** is **tested**.
- Verifying that the **expected results** include as many **variations** and **combinations** as possible and feasible.
- **Boundary** condition.

Test Procedure Design

Test Data Requirements / White-Box Test Data Definition

White-Box (Development-Level) Test Data Definition

TP Number	Design Component	Data Requirement	Description
1530	TV1016	Database tables	Screen inputs
1531	TV1016	Variable input	Range of data values (see test requirement)
1532	TV1016	Variable input	Range of data values (see test requirement)
1533	TV1017	Data object	Requires a bitmapped data object
1534	TV1017	Variable input	Range of data values (see test requirement)
1535	TV1018	Database tables	Screen inputs
1536	TV1018	—	Printer output test using existing data
1537	TV1019	Database tables	Screen inputs
1538	TV1019	Data object	Requires a bitmapped data object
1539	TV1019	Variable input	Range of data values (see test requirement)
.	.	.	.

Test Procedure Design

Test Data Requirements / White-Box Test Data Definition

- Keep in mind **design-related issues,** such as the use of **arrays**, **pointers**, **memory allocations, and decision endpoints.**
- Be **cognizant** of possible **sources of sample data.**
- Source documentation may consist of the following items:
  - Flow graphs (cyclomatic complexity)
  - Data models
  - Program analyzers
  - Design documents, such as structure charts, decision tables, and action diagrams

Test Procedure Design

Test Data Requirements / White-Box Test Data Definition

- **Source documentation may consist of the following items:**
  - Detail function and system specifications.
  - Data flow diagrams.
  - Data dictionaries, which include data structures, data models, edit criteria, ranges, and domains.
  - Detailed designs, which specify arrays, networking, memory allocation, data/program structure, and decision endpoints.

Test Procedure Design

Test Data Requirements / Black-Box Test Data Definition

A review of test data requirements should address **several data concern:**

- **Depth** - volume or size of databases
- **Breadth** - variation of data values and data value categories
- **Scope** - the accuracy and completeness of the data
- **Test execution data integrity** - the ability to maintain data integrity
- **Conditions** - the ability to store particular data conditions

Test Procedure Design

Test Data Requirements / Black-Box (System-Level) Test Data Definition

TP Number	Design Component	Data Requirement	Description
2330	TV1016	Database tables	Screen inputs
2331	TV1016	Variable input	Range of data values (see test requirement)
2332	TV1016	Variable input	Range of data values (see test requirement)
2333	TV1017	Data object	Requires a bitmapped data object
2334	TV1017	Variable input	Range of data values (see test requirement)
2335	TV1018	Database tables	Screen inputs
2336	TV1018	—	Printer output test using existing data
2337	TV1019	Data object	Requires a bitmapped data object
2338	TV1019	Variable input	Range of data values (see test requirement)
2339	TV1019	Database tables	Screen inputs
.	.	.	.

Test Procedure Design

Test Data Requirements / White-Box Test Data Definition

Such source documents may include the following items:

- **System concept documents** (business proposals, mission statements, concept of operations documents)
- **System requirements documentation** (customer requirements definition, system/software specifications)
- **Business rules** (functional/business rule documentation)
- **Entity relationship diagrams** (and other system design documentation)
- **Use case scenarios and data flow diagrams** (and other business process documentation)

Test Procedure Design

Test Data Requirements / White-Box Test Data Definition

Such source documents may include the following items:

- **Event partitioning** (and state transition diagrams)
- **Data dictionaries** (and data element and interface standards, application programming interface documentation)
- **Help desk logs** (pertaining to existing operational systems)
- **User expertise** (end-user input)
- **Regulations and standards** (industry and corporate standards)
- **White-box data definition documentation** (developed by test team)

Test Procedure Design

Test Data Requirements / White-Box Test Data Definition

Test Data Generators

- Automatically generate data
- Derived from specifications or database documentation or manually modified.

Test Procedure (Case) Generators

- This generation is automatic, test procedures can be created as soon as application requirements are recorded.

Chapter summary

- ★ An effective test program incorporating the automation of software testing involves a mini-development life cycle of its own, complete with strategy and goal planning, test requirements definition, analysis, design, and coding.
- ★ Similar to the process followed in software application development, test requirements must be specified before a test design is constructed.
- ★ Much as in a software development effort, the test program must be mapped out and consciously designed to ensure that the test activities performed represent the most efficient and effective tests for the target system. Test program resources are limited, yet ways of testing the system are endless.

Chapter summary

- ★ Following test analysis, the test team develops the test program design models. The first of these design models, the test program model, consists of a graphic illustration that depicts the scope of the test program.
- ★ Having defined a test program model, the test team constructs a test architecture, which depicts the structure of the test program and defines the organization of test procedures.
- ★ The structure of the test program (test architecture) is commonly portrayed in two different ways - test procedure organization method and technique-based test architecture.

Chapter summary

- ★ White-box test techniques are aimed at exercising the software program's internal workings, while black-box techniques generally compare the application's behavior with requirements via established, public interface.
- ★ It is beneficial to be familiar with the kinds of test tools that are available to support the development and performance of related test procedures.
- ★ The design exercise involves the organization of test procedures into logical groups and the definition of a naming convention for the suite of test procedures.

Chapter summary

- ★ At the system level, it may be worthwhile to develop a detailed test design for sophisticated tests. These tests might include test procedures that perform complex algorithms, procedures that consist of both manual and automated steps, and test programming scripts that are modified for use in multiple test procedures.
- ★ Detailed test design may take the form of test program pseudocode, when test programming is required.
- ★ After the detailed test design is complete, test data requirements need to be mapped against the defined test procedures.

References

[1] Book examples include Boris Beizer's *Software Testing Techniques* and John Joseph Chilenski's *Applicability of Modified Condition/Decision Coverage to Software Testing*, to mention only a few.

[2] *Software Considerations in Airborne Systems and Equipment Certification.* RTCA SC-167, EUROCAE WG-12, Washington, DC: RTCA, 1992.

[3] Myers, G.J. *The Art of Software Testing*. New York: John Wiley and Sons, 1979.

[4] Ibid.

References

[5] Ntafos, S. “A Comparison of Some Structural Testing Strategies.” *IEEE Transactions on Software Engineering* 1988; 14:868–874.

[6] Beizer, B. *Software Testing Techniques*, 2nd ed. New York: Van Nostrand Reinhold, 1990.

[7] Adapted from Myers, G.J. *The Art of Software Testing.* New York: John Wiley and Sons, 1979.

[8] Adapted from Myers, G.J. *The Art of Software Testing*. New York: John Wiley and Sons, 1979.

[9] Adapted from SQA Suite Process, January 1996. (See www.rational.com.)