BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING
MASTER OF SCIENCE IN COMPUTER ENGINEERING

RECENT STANDARDS IN SOFTWARE TESTING AND AN APPLICATION

BY

BUĞRA SOYER

MASTER OF SCIENCE THESIS

ANKARA – 2022

BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING
MASTER OF SCIENCE IN COMPUTER ENGINEERING

RECENT STANDARDS IN SOFTWARE TESTING AND AN APPLICATION

BY

BUĞRA SOYER

MASTER OF SCIENCE THESIS

ADVISOR

PROF. DR. AHMET ZİYA AKTAŞ

ANKARA - 2022

BAŞKENT UNIVERSITY

INSTITUTE OF SCIENCE AND ENGINEERING

This study, which was prepared by Buğra SOYER for the program of Computer Engineering, has been approved in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in the Computer Engineering Department by the following committee.

Date of Thesis Defense: 03 / 01 / 2022

Thesis Title: Recent Standards in Software Testing and an Application

Examining Committee Members		**Signature**
Prof. Dr. Ahmet Ziya AKTAŞ, Başkent University	(Advisor)	…………
Assoc. Prof. İhsan Tolga MEDENİ, Yıldırım Beyazıt University	(Member)	…………
Assist. Prof. Tülin ERÇELEBİ AYYILDIZ, Başkent University	(Member)	…………

APPROVAL

Prof. Dr. Faruk ELALDI

Director, Institute of Science and Engineering

Date: … / … / ….…..

BAŞKENT ÜNİVERSİTESİ

FEN BİLİMLER ENSTİTÜSÜ

YÜKSEK LİSANS TEZ ÇALIŞMASI ORİJİNALLİK RAPORU

Tarih: 07 / 01 / 2022

Öğrencinin Adı, Soyadı: Buğra SOYER

Öğrencinin Numarası: 21910235

Anabilim Dalı: Bilgisayar Mühendisliği Anabilim Dalı

Programı: Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı

Danışmanın Unvanı/Adı, Soyadı: Prof. Dr. Ahmet Ziya AKTAŞ

Tez Başlığı: Recent Standards in Software Testing and an Application

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 81 sayfalık kısmına ilişkin, 07/01/2022 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 18'dir. Uygulanan filtrelemeler:

1. Kaynakça hariç
2. Alıntılar hariç
3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları hariç

"Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını" inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrenci İmzası: ………………….

ONAY

Tarih: 07 / 01 / 2022

Prof. Dr. Ahmet Ziya AKTAŞ

………………………………..

Bu tezi yazılım sektöründe yıllarını büyük bir fedakârlıkla geçiren ve ülkemizin yazılım sektöründe ilerlemesine katkı sağlayan tüm yazılım test mühendislerine ithaf ediyorum.

Buğra SOYER

Ankara - 2022

ACKNOWLEDGEMENTS

Yüksek Lisans tez çalışmalarım boyunca akademik ve tez danışmanım olan çok kıymetli hocam Sayın Prof. Dr. A. Ziya AKTAŞ'a göstermiş olduğu desteği, yol göstericiliği ve emekleri için en içten teşekkürlerimi sunarım.

Hayatımın her anında ve tüm eğitim hayatım boyunca her daim desteğini ve fedakârlığını gösteren, bütün kararlarımda yanımda olan ve hep bir ileriki adım için beni motive eden annem Mukaddes SOYER'e ve babam İhsan SOYER'e bütün kalbimle teşekkürlerimi sunarım.

Hiçbir koşulda benden desteğini esirgemeyen değerli kuzenlerim Mert DOĞAN'a, İrem Gökay ÇEVİK'e ve ismini sayamadığım diğer yakınlarıma ve arkadaşlarıma teşekkürlerimi sunarım.

Yıllar boyunca hayatımın her alanında olduğu gibi tez çalışmamın da her aşamasında yanımda olan, görüşlerini, destek ve yardımlarını esirgemeyen değerli arkadaşım Gizem HIZARCIOĞLU'na en içten teşekkür ve sevgilerimi sunarım.

ABSTRACT

Buğra SOYER
RECENT STANDARDS IN SOFTWARE TESTING AND AN APPLICATION
Başkent University Institute of Science and Engineering
Computer Engineering Department
2022

The major objective of this thesis is to study the techniques and methodologies that have been used in practice for Software Testing and understand their relevance to existing international standards.

During the literature survey phase of the study a package of standards, namely ISO/IEC/IEEE 29119 Standard, was discovered. Six parts of this standard covers various topics such as Concepts and Definitions, Test Processes, Test Documentation, Test Techniques, Keyword-driven Testing and finally a Guideline for Software Testing. The last part of this standard was published in July 2021. Thus, it is quite a recent standard in the field of Software Testing. Therefore, an application in Software Testing field may be considered as a valuable contribution using the relevant parts of this standard.

With the aim of a real life application an available open-source Software Requirements Specification (SRS) in Banking Sector is used. During the thesis study, ISO/IEC/IEEE 29119 Standard is applied on that available SRS.

Since an available SRS is used in the application, testing became quite easy.

Extension of the study is also included in the Conclusions Chapter of the thesis.

KEYWORDS: ISO/IEC/IEEE 29119 Standards, Quality in Software, Software Testing Process, Testing in Software.

ÖZET

Buğra SOYER
YAZILIM TESTİNDE GÜNCEL STANDARTLAR VE BİR UYGULAMA
Başkent Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Bölümü
2022

Bu tezin temel amacı, Yazılım Testi için pratikte kullanılan teknikleri ve metodolojileri incelemek ve bunların güncel uluslararası standartlarla olan ilgisini anlamaktır.
Çalışmanın literatür taraması aşamasında, ISO/IEC/IEEE 29119 Standardı adlı bir standart paketi ile karşılaşıldı. Bu standardın altı bölümü, Kavramlar ve Tanımlar, Test Süreçleri, Test Dokümantasyonu, Test Teknikleri, Anahtar Kelimeye Dayalı Test ve son olarak Yazılım Testi Kılavuzu gibi çeşitli konuları kapsamaktadır. Bu standardın son kısmı Temmuz 2021'de yayınlandığı için Yazılım Testi alanında oldukça yeni bir standarttır. Bu nedenle, Yazılım Testi alanındaki bir uygulama, bu standardın ilgili bölümleri kullanılarak oluşturulmuş değerli bir katkı olarak değerlendirilebilir.
Çalışmada gerçek hayata yönelik bir uygulama yapılması amacıyla, Bankacılık Sektöründe mevcut bir açık kaynaklı Yazılım Gereksinimleri Şartnamesi (SRS) kullanılmıştır. Tez çalışması sırasında, mevcut SRS üzerinde ISO/IEC/IEEE 29119 Standardı uygulanmıştır.
Uygulama sürecinde mevcut bir SRS belgesi kullanıldığı için test etme faaliyetleri nispeten kolay bir şekilde uygulanmıştır.
Çalışmanın devamı olabilecek çalışmalara tezin Sonuçlar bölümünde yer verilmiştir.

ANAHTAR KELİMELER: ISO/IEC/IEEE 29119 Standartları, Yazılımda Kalite, Yazılımda Test, Yazılım Test Süreçleri.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT ii

ÖZET iii

TABLE OF CONTENTS iv

LIST OF FIGURES xii

LIST OF TABLES xiv

LIST OF ABBREVIATIONS xv

1. INTRODUCTION 1

1.1 General 1

1.2 Literature Review 2

1.3 Structure of the Thesis 3

2. ISTQB (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD) 4

2.1 General 4

2.2 ISTQB CTFL (Certified Tester Foundation Level) 4

3. ISO STANDARDS (ISO/IEC/IEEE 29119 SOFTWARE AND SYSTEMS ENGINEERING - SOFTWARE TESTING SERIES) 5

3.1 General 5

3.1.1 Part 1: Concepts and definitions 5

3.1.2 Part 2: Test processes 5

3.1.3 Part 3: Test documentation 5

3.1.4 Part 4: Test techniques 6

3.1.5 Part 5: Keyword-driven testing 6

3.1.6 Part 6: Guidelines for the use of ISO/IEC/IEEE 29119 (all parts) in agile projects 6

4. QUALITY DEFINITION IN SOFTWARE DEVELOPMENT 7

4.1 General 7

4.2 Software Quality 7

4.3 Quality Factors 8

4.4 Cost of Quality 8

4.5 Verification & Validation 9

4.6 Prevention of Errors in Early Stages 10

4.6.1 Errors and defects 10

4.6.2 Failures 10

4.7 Defects, Root Causes, and Effects 11

5. SOFTWARE TESTING PRINCIPLES 12

5.1 General 12

5.2 Seven Software Testing Principles 12

5.2.1 Testing shows the presence of defects, not their absence 12

5.2.2 Exhaustive testing is impossible 13

5.2.3 Early testing saves time and money 13

5.2.4 Defects cluster together 13

5.2.5 Beware of the pesticide paradox 13

5.2.6 Testing is context-dependent 13

5.2.7 Absence-of-errors is a fallacy 13

5.3 Software Development Life Cycle Models 14

5.4 Software Development and Software Testing 14

5.5 The Big Picture of Software Testing Strategy 16

5.6 Test Strategies for Traditional Software 16
5.7 Software Development Life Cycle Models In a Project Context 17
5.8 Risks and Types of Risks 17
5.8.1 Product and project risk 18
5.8.2 Quality assurance and risks 18
6. SOFTWARE TESTING IN AN ORGANIZATIONAL AND PROJECT CONTEXT 19
6.1 General 19
6.2 Organizational Context 19
6.3 Project Context 24
7. TEST PROCESSES 27
7.1 General 27
7.2 Organizational Test Process 27
7.3 Test Management Processes 29
7.4 Test Planning Process 31
7.5 Test Monitoring and Control Process 32
7.6 Test Completion Process 32
7.7 Dynamic Test Processes 35
7.8 Test Design & Implementation Process 38
7.9 Test Environment Set-Up & Maintenance Process 38
7.10 Test Execution Process 39
7.11 Test Incident Reporting Process 41
7.12 Static Testing 41
7.13 Differences between Static & Dynamic Testing 42
7.14 Review Processes 42

7.15 Work Product Review Process 43

7.15.1 Planning..43

7.15.2 Initiate review 43

7.15.3 Individual review...43

7.15.4 Communication and analysis of the issues 43

7.15.5 Fixing errors and reporting...44

7.16 Roles & Responsibilities in a Formal Review...44

7.16.1 Author...44

7.16.2 Administration..44

7.16.3 Moderator 45

7.16.4 Review leader...45

7.16.5 Reviewers 45

7.16.6 Scriber 45

7.17 Review Types..45

7.17.1 Informal review 45

7.17.2 Walkthrough...46

7.17.3 Technical review...46

7.17.4 Inspection 46

7.18 Applying Review Techniques...47

7.18.1 Ad-hoc...47

7.18.2 Checklist-based...48

7.18.3 Scenarios and dry runs 48

7.18.4 Perspective-based 48

7.18.5 Role-based 49

7.19 Test Activities & Tasks..49

7.19.1 Test planning 50

7.19.2 Test monitoring and control 50

7.19.3 Test analysis 50

7.19.4 Test design 50

7.19.5 Test implementation 51

7.19.6 Test execution 51

7.19.7 Test completion 51

8. TEST TECHNIQUES 52

8.1 General 52

8.2 Test Types 52

8.3 Functional Testing 54

8.4 Non-functional Testing 54

8.5 Change-related Testing 54

8.5.1 Confirmation testing 54

8.5.2 Regression testing 54

8.6 Black-box Test Techniques 55

8.6.1 Equivalence partitioning 55

8.6.2 Boundary value analysis 56

8.6.3 Decision table testing 57

8.6.4 State transition testing 57

8.6.5 Use case testing 57

8.7 White-box Test Techniques 57

8.7.1 Statement coverage testing 58

8.7.2 Decision coverage testing 58

8.7.3 The importance of statement and decision testing 59

8.8 Experience-based Test Techniques 59

8.8.1 Error guessing 59

8.8.2 Exploratory testing..60

8.8.3 Checklist-based testing 60

8.9 Unit Testing 60

8.10 Integration Testing 62

8.10.1 Top-down integration...62

8.10.2 Bottom-up integration..63

8.10.3 Integration testing levels 63

8.10.4 Integration test work products...63

8.11 System Testing..64

8.12 Quality-Related Types of Testing...64

8.12.1 Accessibility testing 64

8.12.2 Compatibility testing..64

8.12.3 Backup/recovery testing...64

8.12.4 Disaster recovery testing..65

8.12.5 Deployment testing 65

8.12.6 Installability testing..65

8.12.7 Interoperability testing 66

8.12.8 Localization testing..66

8.12.9 Maintainability testing...66

8.12.10 Portability testing 66

8.12.11 Performance testing 67

8.12.12 Stress testing 67

8.12.13 Security testing 67

8.12.14 Usability testing 67

8.12.15 Smoke testing 68

8.13 Acceptance Testing 68

8.13.1 User acceptance testing 69

8.13.2 Operational acceptance testing 69

8.13.3 Contractual & regulatory acceptance testing 69

8.13.4 Alpha & Beta testing 70

9. AN APPLICATION 71

9.1 General 71

9.2 Scope of Testing 71

9.3 Features Not To Be Tested 75

9.4 Test Type 75

9.5 Risk and lssues 75

9.6 Quality Objective 75

9.7 Roles and Responsibilities 77

9.8 Test Methodology 77

9.9 Test Levels 77

9.10 Bug Triage 77

9.11 Resource & Environment Needs 78

9.12 Suspension Criteria and Resumption Requirements 79

9.13 Test Completeness 79

9.14 Test Deliverables 79

9.14.1 Before testing phase 79

9.14.2 During the testing phase 79

9.14.3 After the testing cycle is over 79

9.15 Test Environment 80

10. CONCLUSIONS 81

REFERENCES..82

LIST OF FIGURES

Figure 4.1: Relative cost of correcting errors and defects 9

Figure 4.2: Hierarchy of Verification and Validation activities 11

Figure 5.1: Testing strategy..16

Figure 5.2: Software testing steps 17

Figure 6.1: Multi-layered test context diagram [1] 21

Figure 6.2: Relationship between generic test sub-process, levels and types [1] 22

Figure 6.3: A taxonomy for advanced syllabus test techniques......................................23

Figure 6.4: The multi-layer relationship between test processes 24

Figure 6.5: The test management processes..25

Figure 6.6: Dynamic test processes...26

Figure 6.7: Relationship between the overall project and the test project 26

Figure 7.1: The multi-layer model showing all test processes..28

Figure 7.2: Example Organizational Test Process implementation 29

Figure 7.3: Organizational Test Process 30

Figure 7.4: Example test management process relationships [6]....................................31

Figure 7.5: Test Planning Process [6] 33

Figure 7.6: Test monitoring and control process [6]...34

Figure 7.7: Test completion process [6]..36

Figure 7.8: Dynamic test processes [6] 37

Figure 7.9: Test Design and Implementation Process [6] 39

Figure 7.10: Test Environment Set-Up & Maintenance process [6]...............................40

Figure 7.11: Test Execution process [6] 40

Figure 7.12: Test Incident Reporting process [6]..41

Figure 8.1: Product quality model...53

Figure 8.2: Product quality model (cont'd) 53

Figure 8.3: Unit test 61

Figure 8.4: Unit test environment 62

LIST OF TABLES

Table 9.1: Change Log 71

Table 9.2: Description of the modules (functional requirements) 72

Table 9.3: Software Requirement Specifications 73

Table 9.4: Risk and Issues 76

Table 9.5: Resource & Environment Needs 78

LIST OF ABBREVIATIONS

ATA	advanced test analyst
ATTA	advanced technical test analyst
AUT	application under test
BDD	behavior driven-development
BVA	boundary value analysis
COTS	commercial off-the-shelf
CPU	central processing unit
CTFL	certified tester foundation level
GB	gigabyte
GHz	gigahertz
IEC	international electro-technical commission
IEEE	institute of electrical and electronics engineers
ISO	international organization for standardization
ISTQB	international software testing qualifications board
LAN	local area network
Mb/s	megabits or megabytes per second
QA	quality assurance
RAM	random access memory
RUP	rational unified process
SQA	software quality assurance
SRS	software requirements specification
TDD	test driven-development
UI	user interface
UML	unified modelling language
V & V	verification & validation

1. INTRODUCTION

1.1 General

A software product must be of good quality all the process of software development lifecycle. However, software quality evaluation is not easy. Therefore, the need for quality and process management models and standards in the software development process increases day by day. Furthermore, budget and time-outs in software projects, failures, and the fact that the product is faulty or different from what is requested have pushed organizations to manage an effective software quality management process and ensure that this system is continuous. In this thesis, the issue of improving software quality, primarily through testing, is discussed.

Software is a product that performs a defined function, which has inputs and outputs, runs on any hardware, and contains computer programs and documents such as analysis and design models, use and maintenance guides. The main goals in software projects are to develop software that meets customer requirements, is free from errors, will strengthen the competitiveness of the customer in the market, and will be completed within the specified budget and time. However, some uncertainties in the software development lead to the failure of the software project, mainly due to the lack of user elicitations, incomplete requirements/specifications, and deviations for the requirements. Considering the main reasons stated, finding the fault in software projects is crucial. Revealing the errors in the early stages is very important in reducing the budget and time costs and fulfilling the concepts such as productivity, accuracy, reliability, usability, and proper maintenance within the concept of software iceberg in a positive way.

Testing is necessary in software development for the following reasons [1]:

- The test items being tested does not always perform as intended;
- The test items being tested must be validated;
- The quality contents are being requested by decision makers.
- Test items under consideration must be verified;
- Throughout the software and system development, the test items must be evaluated.

It must be accepted that it is impossible to produce error-free software. For this reason, it is very important to test the software before delivering it to stakeholders eliminate the risk.

This is quite common which are encounter errors, defects and faults from time to time in software projects. Errors which are caused by person who working in the software development project from time to time cause malfunctions if they are delivered to the end user without being detected in a timely manner. If a defect is not encountered in the delivered software product while the program is being used by the end user, it will not affect its operation. A user encountering a software bug can have serious consequences; for example, a mistake can effect public safety, influence economic issues, and effect environmental issues.

Basic goal of testing is minimizing any residual risk associated with the amount of time the test item. Detecting failures in the test product prior to its release for use also reducing risk of poor product quality to stakeholders are also provided. [1].

1.2 Literature Review

Software Testing and Software Testing Applications have been the subject of many needed studies using different techniques, topics, and solution methods. One may see studies dealing with the sector's needs with different assumptions and perspectives. The size of the type of each software test product, the subject it examines, and the solution methods for software tests differ. In test solutions, it is possible to talk about heuristic test methods and methodology and product-based test methods.

In this thesis, the studies in the literature will continue with the studies that have been the subject of the test methods frequently used in the sector. In addition, another topic will be the work done in line with ISO 29119 Standards and the solution methods offered on software tests. Activities related to detecting and correcting errors at early stages in software projects are carried out through software testing processes. In this context, software testing which are evaluate of program behavior with static and dynamic methods, using a certain number of test cases selected from an infinite set, where the software project does not comply with the expected behavior. According to Black, static means without running code, dynamic means by executing code, limited means sufficient number, selected means suitable test cases, and expected means wake to expected and defined characteristics [2].

Testing is a planned set of actions for fault finding that is a verification method. Static, dynamic, limited, selected, and common terminologies are directly related to software testing methodologies and activities.

The ISTQB “International Software Testing Qualifications Board” guide is frequently included in the thesis due to its international validity in software testing processes and its high validity in the industry [3], [4], [5].

“ISO/IEC/IEEE 29119 Software and Systems Engineering - Software Test Series Standards” are included in the relevant sections in the thesis [1], [6 – 10].

1.3 Structure of the Thesis

In the introduction Chapter, which is the first part of this thesis, the purpose of the thesis is explained. Then in the second Chapter of the thesis, the CTFL “Certified Tester Foundation Level” guide published by ISTQB “International Software Testing Qualifications Board” is summarized. This guide has been included in the thesis work as it is the most helpful guide accepted in the software testing world. The third Chapter of the thesis summarizes the “ISO/IEC/IEEE 29119 Software and Systems Engineering - Software Testing Series Standards” Part 1 to Part 6. In the fourth Chapter of the thesis, software quality and testing are discussed in general, and the risks encountered in the software product are mentioned. In the fifth Chapter of the thesis, software testing principles are outlined. In the sixth Chapter, “Software Testing in an Organizational and Project Context” is discussed. In the seventh Chapter, Test Processes are explained in detail. In the eighth Chapter, Test Techniques are discussed. In the ninth Chapter, the topics covered in the thesis are explained on a sample application. At the end of the thesis, related resources, references are presented, and also conclusions of the thesis are given.

The abbreviations in the thesis are listed in the "List of Abbreviations" on page xv.

2. ISTQB (INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD)

2.1 General

"ISTQB", named "International Software Testing Qualifications Board," is an international certification board. The ISTQB, founded in "Edinburgh" in "November 2002", is a non-profit organization officially recognized "in Belgium". "The ISTQB membership includes sixty-six National Testing Boards as well as a handful of internationally known Examination Providers. Furthermore, the ISTQB Industry Partnership Program includes about two hundred fifty members and a network of over three hundred Accredited Training Providers. ISTQB has conducted over one million tests and granted over seven hundred fifty thousand certificates in over one hundred thirty countries through its broad network of Accredited Training Providers, Member Boards, and Exam Providers." The "ISTQB Certified Tester" certificate that software testers. The Certified Tester Foundation level provides the foundation for acquiring additional skills and certification through "Test Analyst or Test Management, Agile Tester and Agile Test Leadership, or into more specialized fields of testing (e.g., Test Automation, Security Testing, and Performance)." [3], [4].

2.2 ISTQB CTFL (Certified Tester Foundation Level)

The ISTQB CTFL, also known as the ISTQB Foundation Level, is the initial stage towards obtaining ISTQB software testing certification. The ISTQB CTFL test is required for all other ISTQB certifications, which means one must have completed the ISTQB Foundation Level exam before taking any other ISTQB certification exam. The ISTQB CTFL is useful for software testers since it builds a foundation of software testing terminology and ideas. In addition, companies value ISTQB CTFL because it improves communication and efficiency inside and between their testing teams, whether in-house or outsourced [5].

3. ISO STANDARDS (ISO/IEC/IEEE 29119 SOFTWARE AND SYSTEMS ENGINEERING - SOFTWARE TESTING SERIES)

3.1 General

Software testing is covered under “ISO/IEC/IEEE 29119 Software and systems engineering - Software testing”, consisting of five international standards and a software testing guideline. The “ISO/IEC/IEEE 29119” family of software testing standards were designed to produce an internationally agreed-upon set of software testing standards that any firm may use while performing software testing. The standard, created in 2007 and released in 2013, “defines vocabulary, methods, documentation, methodologies, and a process evaluation model for testing that may be utilized across the software development lifecycle.” [1].

3.1.1 Part 1: Concepts and definitions

“ISO/IEC/IEEE 29119-1” standard simplifies the use of the other “ISO/IEC/IEEE 29119” standards by defining the ideas and vocabulary that these standards are founded on and providing examples of their use in practice. The “ISO/IEC/IEEE 29119-1” standard is instructional, acting as a foundation, backdrop, and direction for the following parts. [1].

3.1.2 Part 2: Test processes

The “ISO/IEC/IEEE 29119-2” [6] standard offers test process descriptions outline of the software testing techniques related to organizational, test management, and dynamic levels. All types of testing are supported, dynamic testing, functional/non-functional testing, manual/automated testing, and also scripted/unscripted testing. Any software development lifecycle model can benefit from the insights presented in this article. Because testing is an essential part of software development risk mitigation, “ISO/IEC/IEEE 29119-2” takes a risk-based approach to testing.

3.1.3 Part 3: Test documentation

“ISO/IEC/IEEE 29119-3” [7] includes templates and examples of test documentation. The templates are divided into clauses that correspond to the basic structure of the

"ISO/IEC/IEEE 29119-2" test process description, the test process in which they are developed. "ISO/IEC/IEEE 29119-3" supports dynamic testing, functional/non-functional testing, manual/automated testing, and also scripted/unscripted testing. The documentation templates given in "ISO/IEC/IEEE 29119-3" [7] can be used with any software development lifecycle model.

3.1.4 Part 4: Test techniques

Many standard test techniques are described in "ISO/IEC/IEEE 29119-4" [8]. They used during the test design and implementation process that are outlined in "ISO/IEC/IEEE 29119-2" [6]. Test cases can be generated using the test methodologies presented in this article. It can be used to gather evidence that test item criteria have been met and that a test item has flaws when performed. Risk-based testing (described in "ISO/IEC/IEEE 29119-1" [1] and "ISO/IEC/IEEE 29119-2" [6]) would be used to establish the set of approaches that apply in various scenarios. This standard permits the creation and execution of test cases at any stage or for any testing.

3.1.5 Part 5: Keyword-driven testing

"Keyword-Driven Testing" is defined by ISO/IEC/IEEE 29119-5 [9] as a modular method of expressing test cases. This standard lays out the architecture for test engineers to communicate test artifacts like test cases, test data, keywords, or full test specifications using Keyword-Driven Testing.

3.1.6 Part 6: Guidelines for the use of ISO/IEC/IEEE 29119 (all parts) in agile projects

This standard explains how to use "ISO/IEC/IEEE 29119 in agile development life cycles (all sections)" [10]. Testers, test managers (qa managers), business analysts, product owners, scrum masters, and developers who work on agile projects to concern about this document. The mappings provided in this article, on the other hand, are meant to assist any team or organization.

4. QUALITY DEFINITION IN SOFTWARE DEVELOPMENT

4.1 General

The degree to which client expectations are satisfied is directly related to the program's quality. Customer expectations are satisfied and to what degree the expectations are relevant due to software testing efforts. This chapter provides general overview of software quality.. Later on, we will go through the link between quality and testing in software development. Computer technologies are an indispensable aspect of one's life, from workplace applications to consumer items. The majority of individuals have come into touch with programs that did not function as expected. If the software program fails to perform as expected, it can result in a loss of revenue, time, or money and an accident or death. Software testing assesses the software's consistency and reduces the risk of software failure while in use. Another common issues regarding testing is that only focuses on confirming standards and user stories. Although this process entails determining if the system fulfills set criteria, validation entails determining whether the system meets the customer's demands and other stakeholders in its working environment.

4.2 Software Quality

A valuable product not only provides the contents and functionalities that the end-user wants, but also error-free manner. A valuable things often follows the criteria that stakeholders have specified clearly. Furthermore, it is expected that all high-quality applications must fulfill several tacit criteria. High-quality software supports both the software organization and the end-user population by providing value to both the manufacturer and the software product user.

The software corporation gains more beneficiation when the software requires less maintenance work and less customer support. As a result of all these, a software product that is created with maximum profitability and efficiency and that serves the stakeholders is created. [11], [3].

4.3 Quality Factors

Pressman and Maxim [11] described functionality as software functionality, suitability, accuracy, interoperability, compatibility, and reliability.

4.4 Cost of Quality

This part includes both the expenses of pursuing quality and the costs of undertaking quality-related activities, as well as the costs incurred as a result of a lack of quality. According to Pressman and Maxim [11] the cost of quality-related three as with prevention, appraisal, and failure.

Prevention costs includes cost of management activities procedures to plan not only quality control but also quality assurance activities, the cost of additional technical activities to develop design models, complete requirements and test planning costs.

The time and money spent planning test activities, preparing test cases and data, and running those test cases once are all part of the appraisal costs. There are two sorts of nonconformance costs: internal failures and external failures. Failure costs would be eliminated if no faults were discovered before selling a product to clients. There are two sorts of failure costs: internal failure costs and external failure costs. When a flaw in a product is found prior to shipment, internal failure costs are incurred.

External failure costs include complaint resolution, product return and replacement, and labor costs are associated. A bad reputation and resultant loss of business is another external cost of failure that is difficult to quantify but very real. Undesired things happen when poor quality software is produced. Many external failure costs, such as goodwill, are difficult to measure and, as a result, many organizations overlook them when evaluating cost-benefit ratios.

Other external failure costs decreased while customer satisfaction is unchanged. As shown in Figure 4.1 modified from [11], the relative costs of locating and correcting a mistake or defect rise considerably as one progresses from prevention to detection to internal failure to external failure [11].

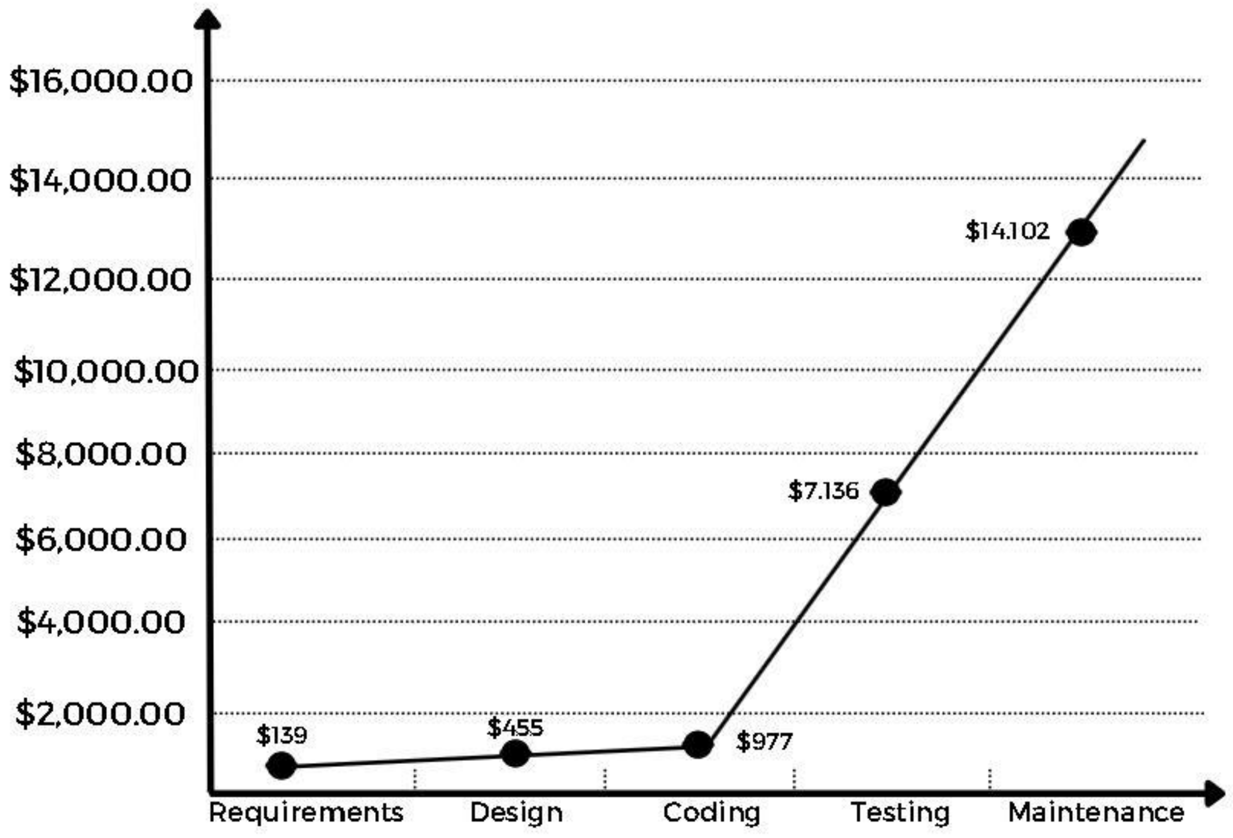


Figure 4.1: Relative cost of correcting errors and defects

4.5 Verification & Validation

According to Pressman and Maxim, software testing is part of a more comprehensive subject known as verification and validation [12].

Validation is a collection of activities that ensures that the software development process meet the needs of customer's requirements.

- Verification: "Are one building the product right?"
- Validation: "Are one building the right product?"

Figure 4.2 modified from [1], defines a verification and validation (V&V) procedures.

Technical audits, quality and configuration audits, performance monitoring, modeling, and feasibility studies, documentation reviews, database reviews, algorithm analysis, development testing, usability testing, qualification testing, acceptance testing, and installation testing are all examples of SQA activities [1], [12], [3].

4.6 Prevention of Errors in Early Stages

In software development activities, it is crucial to find errors in the early stages and while the product is still in the analysis, design, and development stages. These tests, carried out in the early stages and before the project is started, are defined as static tests. Before going into the details of static tests, first look at how the errors, defects, and failures are defined.

An individual can make a mistake, resulting in the implementation of a fault in software code or another job object.

Requirement elicitation error might be cause requirement failure, which could directly related to programming problems, which could then lead to a code defect. Any flaws, for example, to induce a malfunction, require extremely complicated inputs or preconditions, which might occur seldom or never.

4.6.1 Errors and defects

According to ISTQB [3], [4], errors and defects can occur for a variety of reasons, including timing is critical, participants in the initiative are inexperienced or under-skilled, miscommunication between project partners, including specifications and design miscommunication, the code, specification, construction, fundamental problem to be solved, and technology used are all complex, interfaces complexity, and the code, specification, construction, fundamental problem to be solved.

4.6.2 Failures

Failures are caused by various factors, including flaws in the code and environmental factors. Unexpected test outcomes are not all disasters. False positives can happen for various causes, including flaws in the way experiments were run, faults test results, test environments, or other test ware, and so on. Similar mistakes or faults can also lead to false negatives in the opposite case. False negatives are checks that miss defects that should have been found, and false positives are identified as defects but are not.

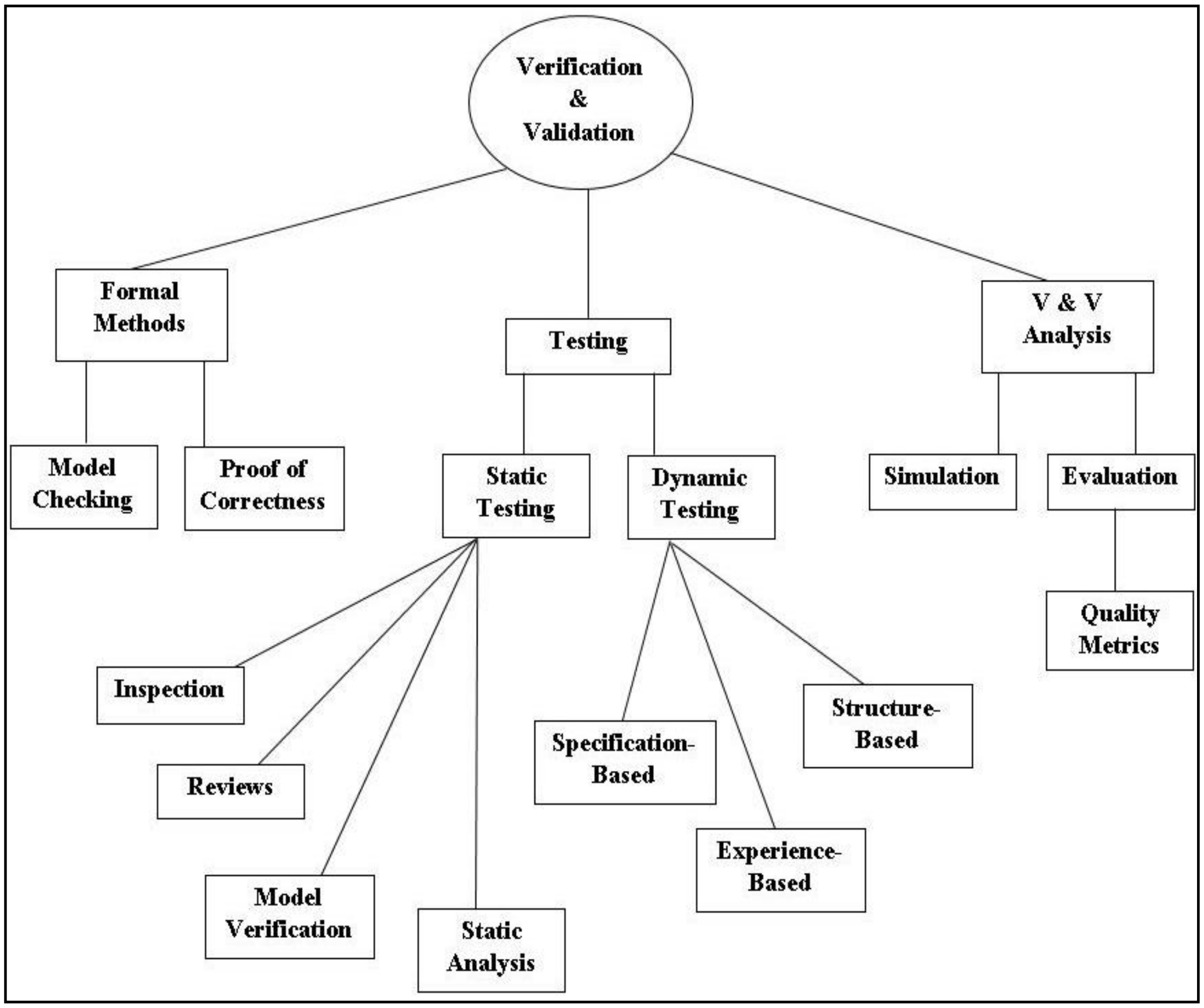


Figure 4.2: Hierarchy of Verification and Validation activities

4.7 Defects, Root Causes, and Effects

This subject is discussed and explained in detail through an example case. The product owner misunderstood how to measure interest. The faulty code was written for an unclear user story. Suppose there are many flaws in interest calculations, and similar misunderstandings cause these defects. In that case, product owners should be educated on the subject of interest calculations in order to avoid potential defects.

The inaccurate measurement in the code is a flaw, and it arose from the user story's original flaw of complexity. The original flaw was due to the product owner's lack of expertise, resulting in a user story mistake [3], [4].

5. SOFTWARE TESTING PRINCIPLES

5.1 General

Basic aim of testing is to find mistakes, and suitable test has a high chance of doing so. As a result, while designing and implementing a computer-related system, one should keep testability in mind.

All potential outputs may be formed by combining inputs, and input - output formats are consistent and well organized. Controlling the scope of testing allows one to pinpoint problems faster and undertake better retesting. The software system is composed of self-contained components that may be tested individually. Changes to the program are few, well-managed, and do not invalidate current tests. As a result, the program recovers quickly from errors.

An appropriate test has a high likelihood of detecting a mistake. The tester should comprehend the software and create a mental image of how the software can fail. There is no value in running a test that serves the same goal as another. Each test should serve a distinct function.

5.2 Seven Software Testing Principles

A series of research standards have been proposed that include essential guidance for all testing. These standards show that many types of testing principles are directly related to software quality and testing.

The seven principles shown and explained below are related to each other, and when any of them are violated, it can cause problems with a quality test procedure over time [3], [4].

5.2.1 Testing shows the presence of defects, not their absence

Test process will reveal whether or not faults exist, but it cannot ensure that there are not any.

Checking decreases the likelihood of undiscovered bugs existing in the program, but it is not proof of correctness even though no defects are identified.

5.2.2 Exhaustive testing is impossible

Except in the most trivial situations, one hundred percent testing is impossible. Therefore, risk estimation, test methods, and priorities should be used to narrow down research efficacy rather than measuring everything.

5.2.3 Early testing saves time and money

Not only static but also dynamic testing procedures discover problems. Two of the test processes should be started early in the process.

5.2.4 Defects cluster together

Most defects found during pre-release testing are usually included in a limited number of components or are to blame for most organizational problems.

Defect clusters are an essential part for risk analysis issues.

5.2.5 Beware of the pesticide paradox

Test processes are running repeatedly, they will eventually stop uncovering new bugs. After that, existing test data and same testing process will need to be updated, and new tests will need to be developed to discover new problems.

In other respects, the pesticide paradox is beneficial, such as automated regression testing, resulting in fewer regression faults.

5.2.6 Testing is context-dependent

Different testing methodologies are required for different contexts. For example, e-commerce application is evaluated differently from safety-critical industrial automation software. Likewise, testing in an agile project varies from testing with a linear product development lifecycle.

5.2.7 Absence-of-errors is a fallacy

Testers will examine all available samples and find problems, although this is unlikely due to the reasons in the first and second clauses. In addition, the absence of errors in the system as a result of the tested methods does not mean that the system works completely correctly.

5.3 Software Development Life Cycle Models

A software development life cycle model or a methodology explains activities that are performed at each part of the software project and how the activities related to each other logically. According to ISTQB, many different software development lifecycle models require different testing approaches [3], [4].

5.4 Software Development and Software Testing

There are many types of characteristics occur in appropriate testing practice in every software development model [3], [4]:

Every software development activity has a related test activity. During the appropriate software development activity, test analysis and design for a particular test level begin, testers take part in conversations to discover and enhance requirements and design, and they analyze work products as soon as drafts are ready, testing activities should begin early in the life cycle and follow early testing guidelines.

A sequential software development model, such as waterfall method model defines sequence of activities. This means software development process must start after the previous one has been completed.

In the waterfall paradigm, software development tasks are completed one after the other.

Testing is only performed when other software development activities under this paradigm have been completed.

Unlike the waterfall paradigm, the V-model includes tests and follows the concept of early testing throughout the software development process. Also, the V-model includes test levels for each software development step that aids early testing. In this architecture, tests associated with each test level are executed sequentially. However, overlaps do occur in some cases. Sequential software development models aim to share the software with stakeholders and users after a fully-featured software is created, so months or years may be required for the software to go live.

Incremental software development models include identifying requirements in parts. Software features developed step by step. The size of the software features to be covered in these increments may vary as some models have large parts and others smaller parts. It can be as small as a software feature to be addressed, a new query option, or a single change to the UI screen.

According to the ISTQB, Iterative software development means that the software features under consideration are defined, developed, constructed, and tested in several cycles, generally of a fixed period. Iteration may involve modifications to the project's scope and updates to software features established in earlier cycles.

The iterative techniques are summarized [3], [4]:

Rational Unified Process (RUP): Each cycle is typically two to three months long, and software features addressed in cycles often contain two or three sets of software features, as well as associated software features.

Scrum: Each cycle is often brief (a few hours, days, or weeks), and the software features evaluated are similarly limited. In addition, the following topics will go through this sort of technique in further depth.

Kanban: Applied without fixed-length loops; once completed, it can offer a single development or software feature or provide a group of software features required for release at a time.

Spiral: Spiral involves creating visual or experimental software features, some of which may be substantially updated.

Scrum, LAPIS, and Kanban are agile methodologies and are efficient for relatively small projects. RUP and Spiral are suitable for large projects. Software developed using these methods often intersects with test levels throughout development and involves repeated application. In order to proceed to live deployment, each software feature should be evaluated at several levels. According to ISTQB, the methodologies used by many software development projects also show how to organize test work.

These strategies establish a developing system; activation might be based on software features, cycle, or a more traditional major release basis in these ways. Regression testing becomes increasingly critical as software evolves, regardless of whether features are brought live or not [3], [4].

5.5 The Big Picture of Software Testing Strategy

Figure 5.1 modified from [12], shows the software development process as a spiral.

Moving inward around the spiral eventually leads to the design and then the code. To generate computer software, inward spirals are drawn abstraction at each turn [12].

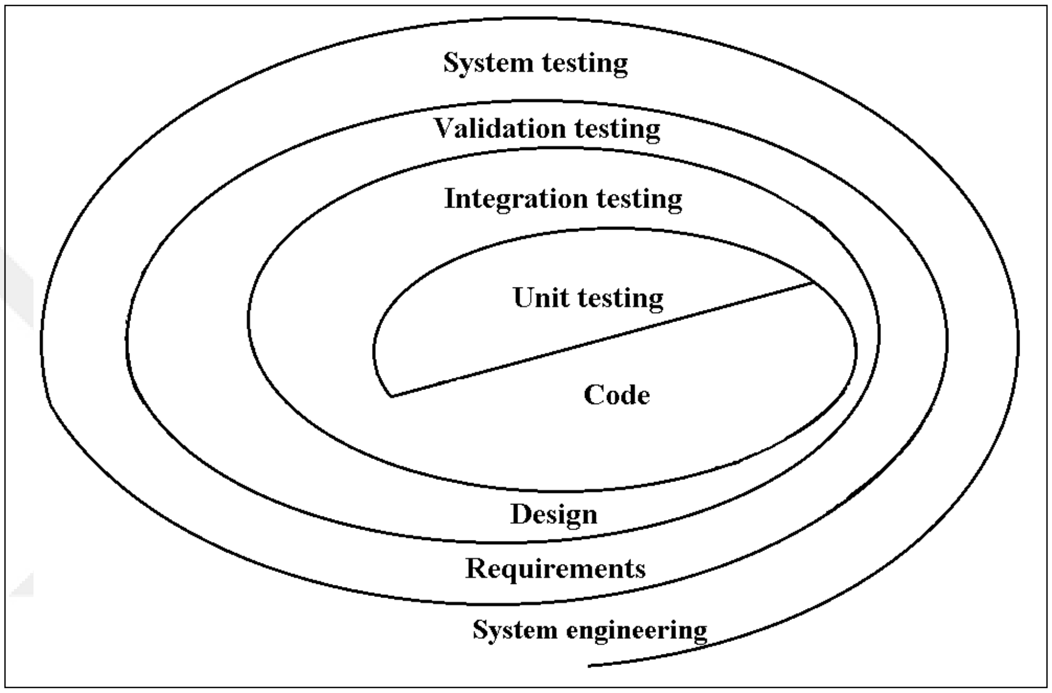


Figure 5.1: Testing strategy

At the last stage of the figure, it comes to system testing. When testing computer software, a spiral is drawn along streamlines that expand the scope of the test at each turn [12]. Figure 5.2 modified from [12], shows that test case design techniques.

However, techniques implement certain programs can be used to ensure that important control paths are covered. The software was integrated, a series of high-grade tests were carried out. Then, finally, the validation criteria should be evaluated [12].

5.6 Test Strategies for Traditional Software

Each level of testing constitutes a subset of the testing process that requires actions related to particular stage of software development process.

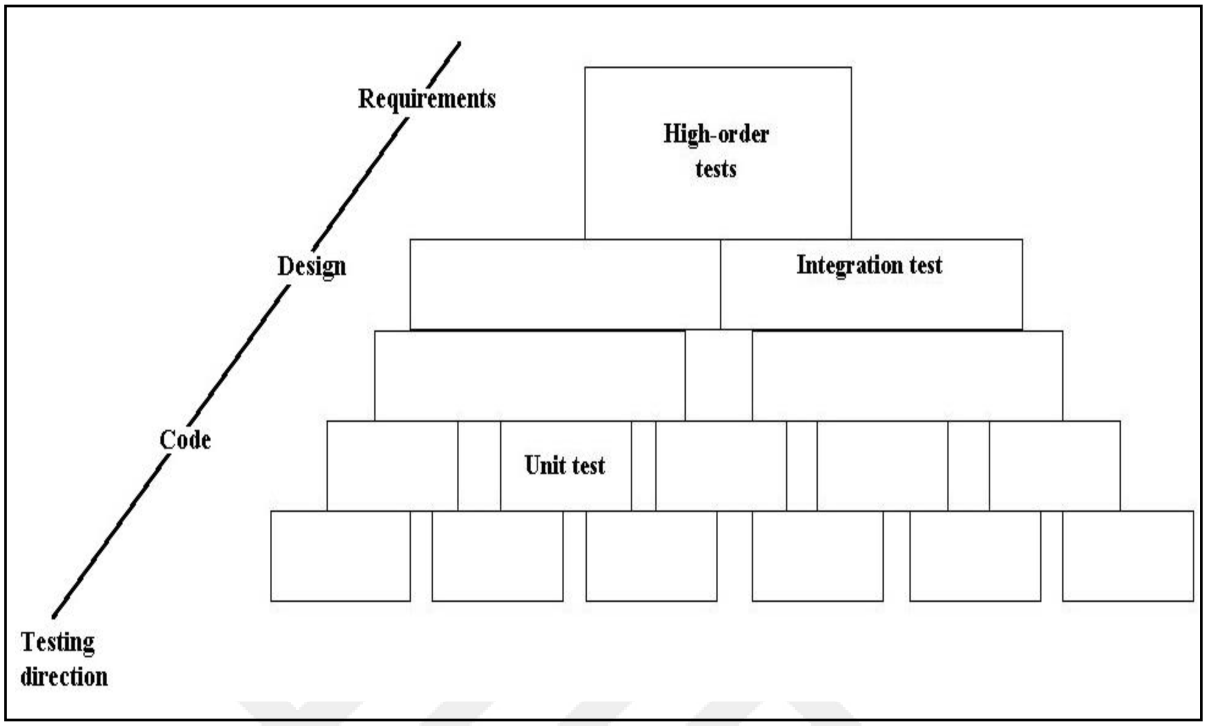


Figure 5.2: Software testing steps

5.7 Software Development Life Cycle Models In a Project Context

Models for software development lifecycles should be selected and modified related to projects' environment and specifications. For example, according to ISTQB, the selected test methods should be different according to the project aim, the product being developed. For example, testing methods of a safety critical system and an online shopping application should be different from each other. Another example is that organizational and cultural issues within the organization can prevent team members from communicating, rendering a circular software development model worthless for software development in that organization. Context of the project may be necessary to combine or reorganize the testing levels and testing activities. For example, interoperability tests at the system integration testing level and acceptance testing level for the integration of commercial applications [3], [4].

5.8 Risks and Types of Risks

The probability of an incident and the effect of that event define the risk level.

5.8.1 Product and project risk

Product risks related to the encounter that business product will fail to meet the needs of customers and stakeholders.

Product risks related to quality risks. When it comes to certain aspects of a product's quality.

Project risks, on the other hand, are related to the development and testing processes that occur within the organization and are at the production stage of the product [3], [4].

5.8.2 Quality assurance and risks

Quality assurance forms the basis of sound software engineering methods. Project management and quality control procedures, all of which are necessary to develop high-quality software.

The aim of quality assurance is to provide manage technical personnel with the information. They need to be informed about product quality and to gain understanding and confidence in the mechanisms available to achieve product quality.

Software quality assurance covers more actions related to software quality management [3], [4].

6. SOFTWARE TESTING IN AN ORGANIZATIONAL AND PROJECT CONTEXT

6.1 General

A particular software or system development project carried out within any software institution needs specific processes tied to the organization's processes. For this reason, businesses that produce or purchase software products engage in software development activities according to their internal structures within the institution.

In addition, testing activities, which are among the essential stages of these activities, have to be similarly dependent on several procedures.

6.2 Organizational Context

Organizations manufacture or purchase software to develop effective, and efficient processes. They often create a comprehensive set of lifecycle protocols for development projects. These protocol kits are intended to help organizations adopt it and use it for custom projects. Within an organization, a single software or systems development project will most likely follow the organization's processes. In some cases, an organization may complete a project without a suitable set of processes.

Software testing should adhere to the most significant degree of organizational management, according to ISO/IEC/IEEE 29119-1 [1]. This commitment should be enshrined in an “Organizational Test Policy” and one or more “Organizational Test Strategies”, which serve as the foundation for internal software testing. Organizational test policies and procedures are only seen in more progressive organizations. Without specific test rules and organizational test procedures, testing is carried out in organizations with a lower level of maturity. This method, however, leads to a lack of consistency and, as a result, makes project testing less effective and efficient.

According to industry experience, no one test strategy, plan, technique, or procedure will work in all scenarios. As a result, organizations and projects should modify and enhance testing details in light of standards like this one. Test activities performed as part of the project should be considered in the overall project plan. “Organizational Testing Policy” and

“Organizational Testing Strategy” and any deviations from organizational principles should be included in Project Testing Plan.

The weighting of diverse test demands and the balance of resources among the various tests is an essential component of test planning. The conclusion of this analysis is documented in the Test Plan.

Each test sub process can have its own Test Plan, which consists of a test sub process strategy aligned with the project's testing strategy, and information specific to the test sub process [1].

The testing function in a multi-tier architecture is built using the UML (Unified Modeling Language) notation in Figure 6.1 [1].

“Organizational Test Policy” is implemented at this level. At this stage, a testing strategy is designed related to context of the project. The Project Test Plan is built around the project schedule and testing strategy.

All testing strategy and testing methodology will be covered in the Project Test Plan. Defines appropriate testing sub-processes and establishes the testing framework for the project, outlining their purpose, procedures, resources, and time frame.

Detected sub-processes are documented in the sub-process test plan. The test plan identifies appropriate design approaches (static or dynamic) to perform the tests required by the sub process plan.

The test execution strategy will be included in the child process test plan (for example, scripted, unscripted, or a mix of both).

Test strategies are suited to the specific environment of the test project. The testing techniques in Figure 6.1 should target the concerns identified in the Multi-layered testing context diagram. Each test plan will be unique, incorporating various combinations of test sub-processes, automation levels, and test design techniques, completion criteria, timing, and sourcing.

Planning and selection begins early in a project when variables such as risk change and continue throughout the test lifecycle. As a result, expect many parts of testing strategy and methodology to change. At the same time, project, organizational and legal constraints will likely limit these changes.

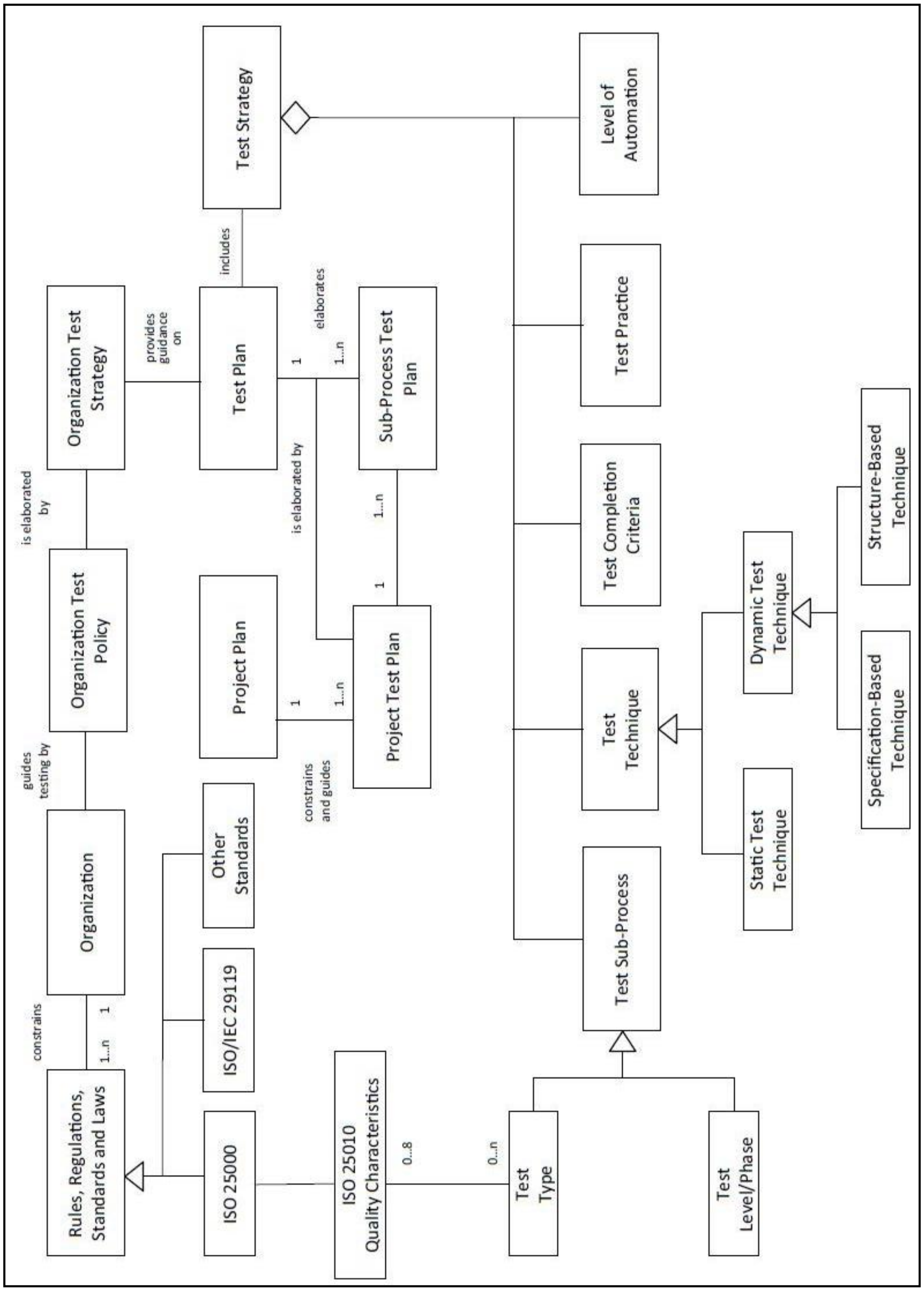


Figure 6.1: Multi-layered test context diagram [1]

The relation between the general testing process, general testing sub-processes, testing levels or phases, and testing types is shown in Figure 6.2 [1]. Additionally, figure shows the deployment of common test sub processes.

In other words, each testing level or stage is a specific implementation of the overall testing sub process.

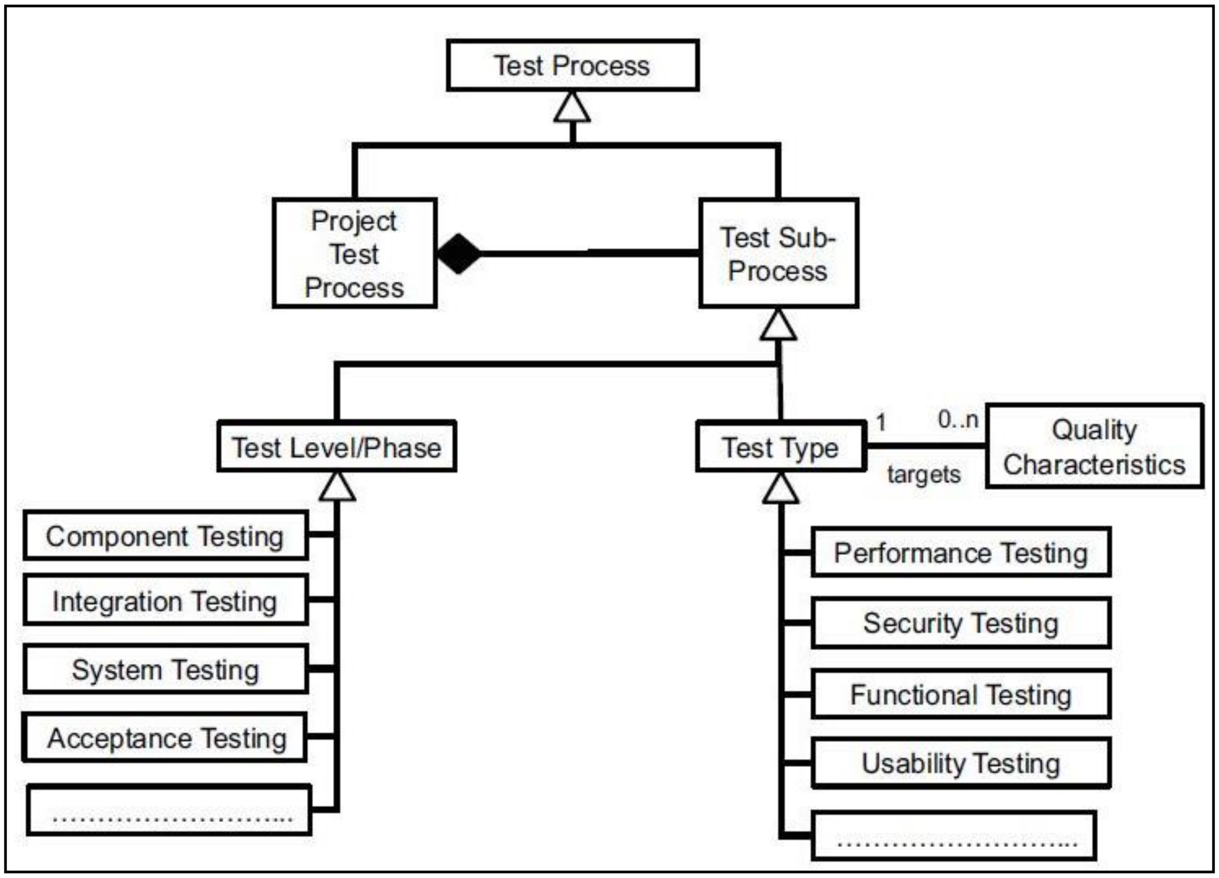


Figure 6.2: Relationship between generic test sub-process, levels and types [1]

Figure 6.3 modified from [2], shows taxonomy for advanced syllabus test techniques. It shows the ultimate breakdown of testing techniques as the distinction between static and dynamic testing. Static tests included in inspections and static analysis. Static analysis is dependent on a tool as key fault finder and inspection expert. Black, divides dynamic tests into five main types, as shown in the figure.

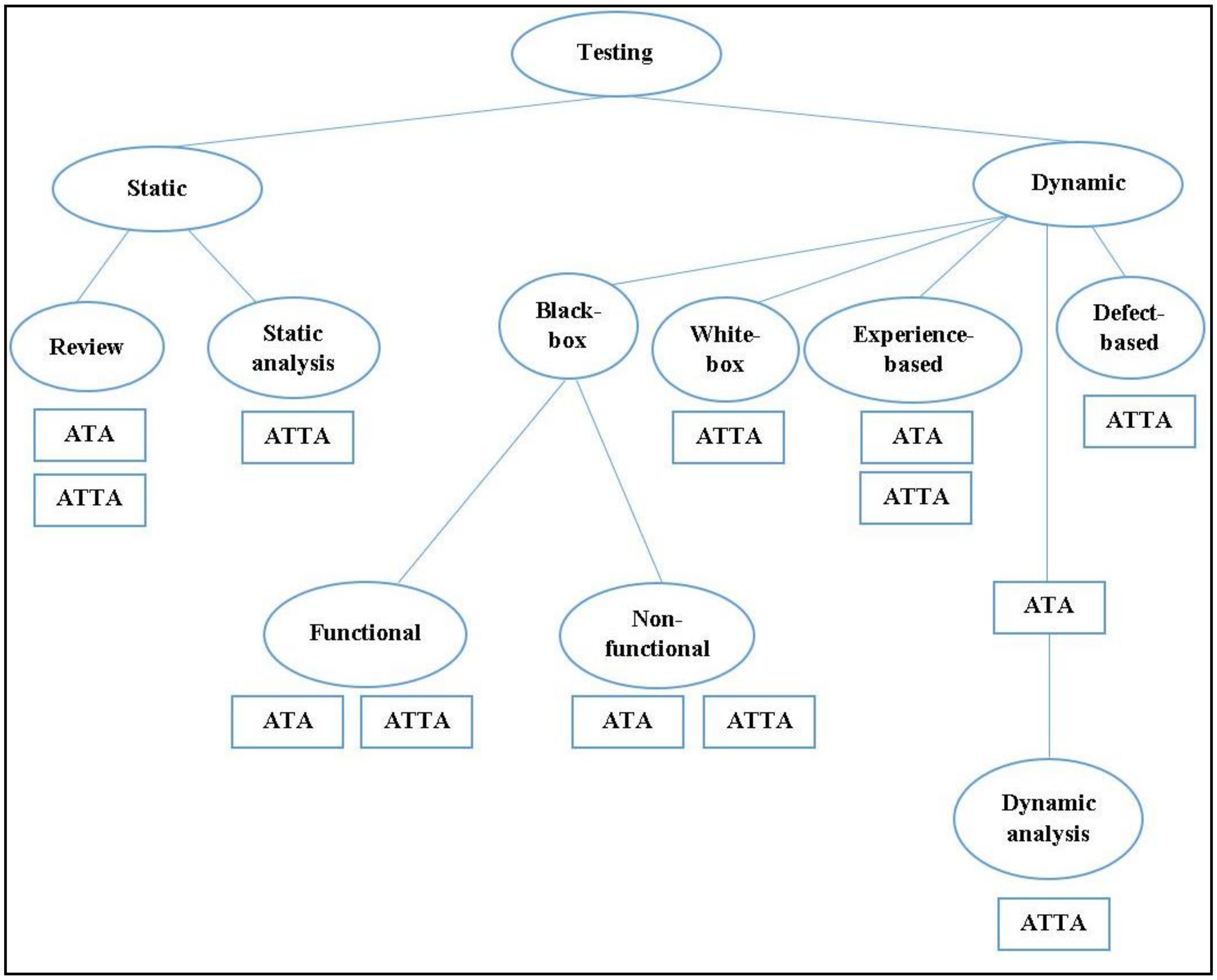


Figure 6.3: A taxonomy for advanced syllabus test techniques

Black-box (also called specification-based or behavioral) test is based on how the system should function. Black-box tests has two types which are functional and non-functional. Functional and non-functional tests are easily distinguished because functional tests address what the system does, whereas non-functional tests address how the system accomplishes what it does. White-box (also called structural) test based on how the system is built. Experience-based test based on the tester's instincts and abilities and the tester's previous experience with similar applications or technologies. Dynamic analysis test is directly related to the tester's examination of a software program while operating, generally using code instrumentation. Finally, defect-based testers construct tests based on their knowledge of the defect addressed by the test with tests developed systematically from what is known about the defect.

6.3 Project Context

The organizational layer in the process model is in charge of high-level test needs such as the “Organizational Test Policy and Organizational Test Strategy”.

The middle layer then moves to test management, while the bottom layer defines multiple dynamic testing test processes. The three-layer process model is depicted in Figure 6.4 modified from [1].

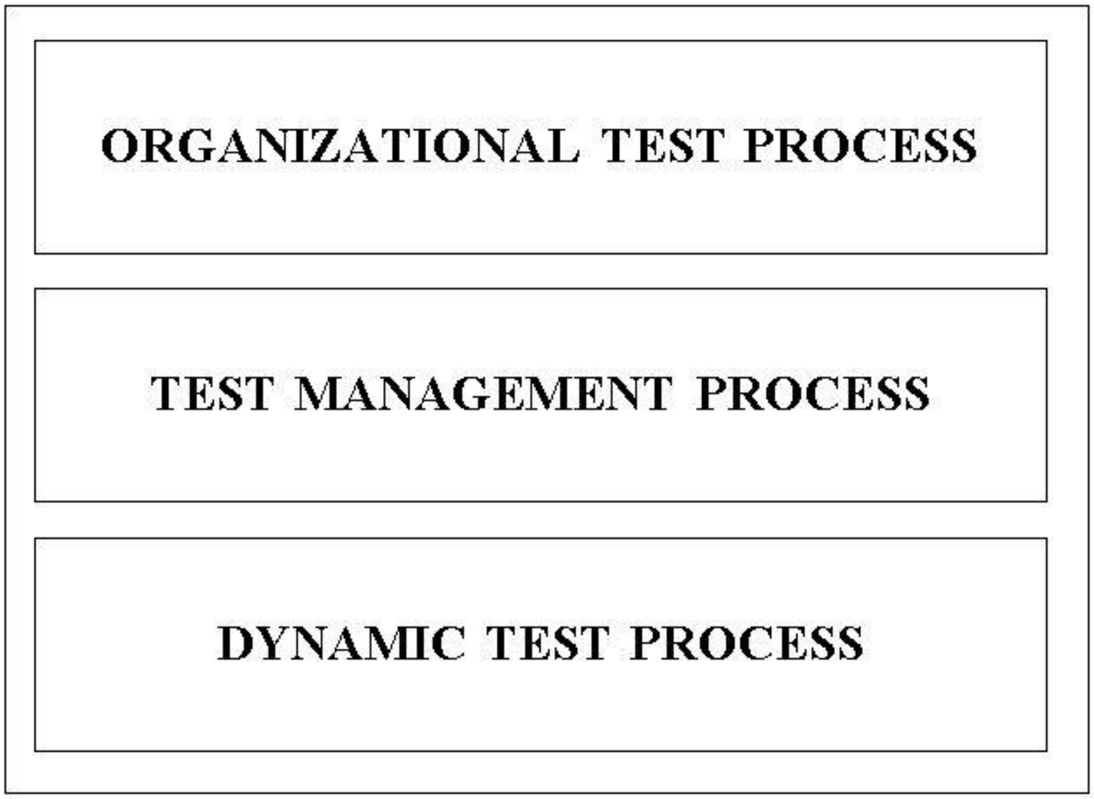


Figure 6.4: The multi-layer relationship between test processes

In business terms, the Test Policy outlines the organization's expectations of the management strategies and methods related to software testing.

The Test Policy directly recommended direction and conduct in designing and implementing the “Organizational Test Strategy” and the organization's test operations.

The “Organizational Test Process” explains how the “Organizational Test Policy” is created, implemented, and maintained. The “Organizational Test Strategy” explain requirements and constraints.

Figure 6.5 modified from [1], defines the management of the testing to be undertaken. A project-related test plan is designed based on examining recognized risks and project restrictions and the “Organizational Test Strategy”. This strategy is created by defining the static and dynamic testing to be performed, the overall staffing, and balancing the specified constraints (resources and time) with the scope and quality of the test job to be completed. This is documented using a “Project Test Plan”. Throughout the testing process, monitoring activities are conducted to testing as planned and that risks are adequately managed. Testing activities are required, control directives sent to the relevant test process or sub-process. Test Status Reports may be prepared frequently during this monitoring and control stage to keep stakeholders informed of test progress. The Project Test Completion Report summarizes the project's overall testing results. A project's extensive testing is typically broken down into smaller sub-processes. They should controlled, carried out, and reported the same manner as the rest of the test project [1].

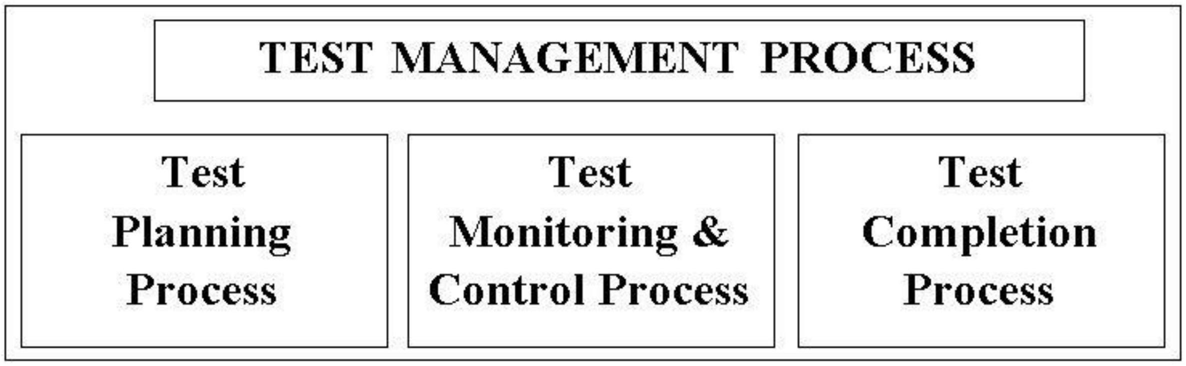


Figure 6.5: The test management processes

Static and dynamic testing may be included in test sub-processes. Figure 6.6 modified from [1], depicts the dynamic test procedures thoroughly discussed in the following sections. The support mechanisms used planning of a project, including the management of the test project which are referred to as project. Regardless of who is in charge of the separate procedures, the project management, and test management processes are inextricably linked, as seen in Figure 6.7 modified from [1].

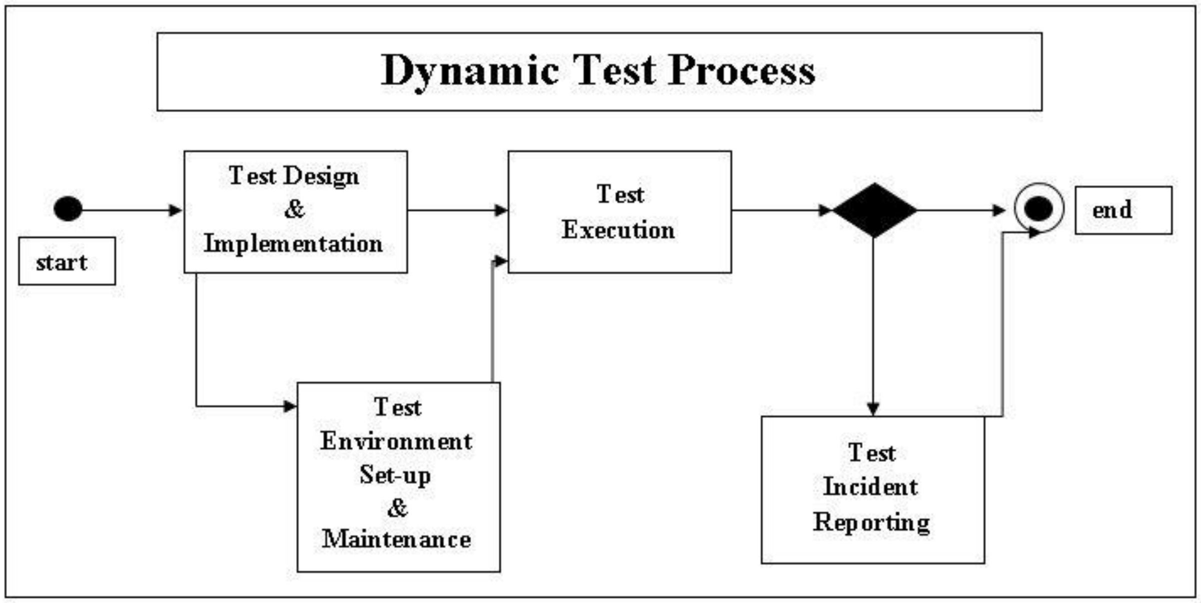

Dynamic Test Process
start
Test Design & Implementation
Test Execution
end
Test Environment Set-up & Maintenance
Test Incident Reporting

Figure 6.6: Dynamic test processes

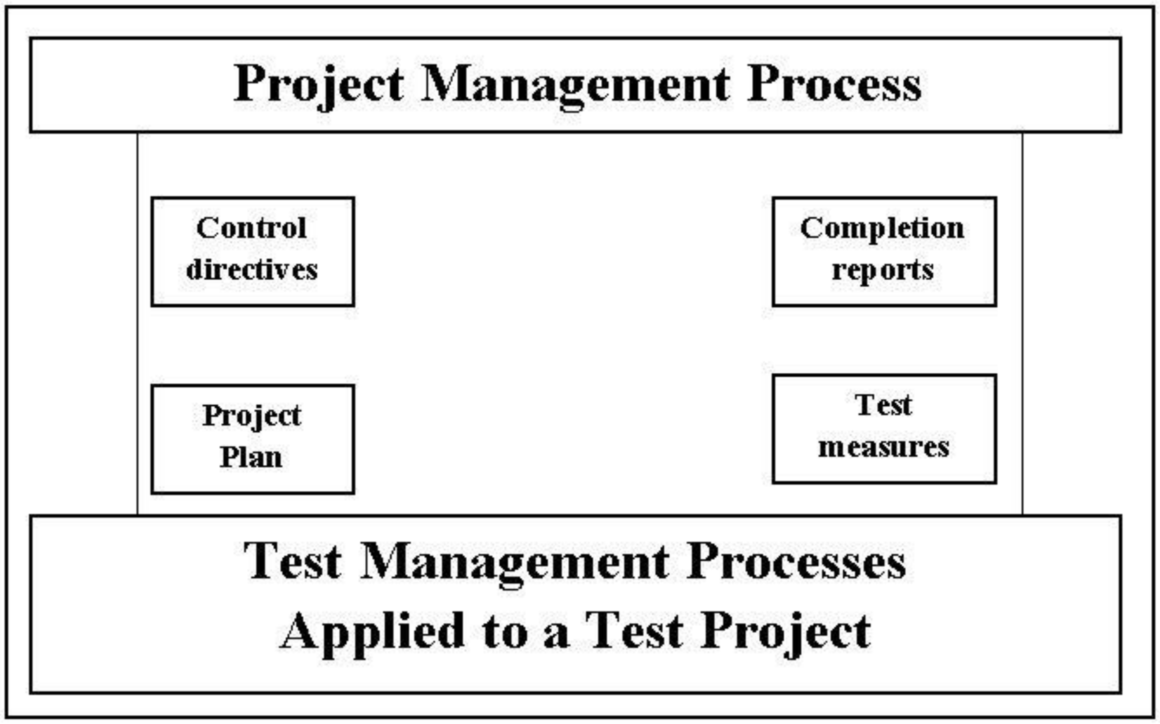

Project Management Process
Control directives
Completion reports
Project Plan
Test measures
Test Management Processes Applied to a Test Project

Figure 6.7: Relationship between the overall project and the test project

7. TEST PROCESSES

7.1 General

"ISO/IEC/IEEE 29119-2" [6] standard groups the testing processes that performed during the life cycle of a software system into three process groups, as described in the multi-layer relationship between test processes (See Figure 6.3). According to "ISO/IEC/IEEE 29119-2" [6], the "Organizational Test Process" creates and maintains specifications such as "Organizational Test Policies", and other assets. "Test Management Processes" defining procedures for managing testing for an entire testing project. They have test planning, test monitoring and control, and test completion process.

Dynamic test processes designing generic processes for dynamic testing. Dynamic testing may be conducted at a specific phase of testing or for a specific type of testing.

Test design and implementation process, test environment set-up and maintenance process, test execution process, and test incident reporting process are the dynamic test processes.

As seen in Figure 7.1 modified from [6], the layers of the test process model contain variable numbers of test processes.

7.2 Organizational Test Process

Organizational test requirements are developed and managed using the "Organizational Test Process". According to ISO/IEC/IEEE 29119-2 [6], these criteria are usually applicable to testing across the whole business. Organizational test standards include the "Organizational Test Policy and the Organizational Test Strategy".

The organizational test process is general and may be used to create and maintain non-project-specific test documentation, such as a Programme Test Strategy that applies to several related projects.

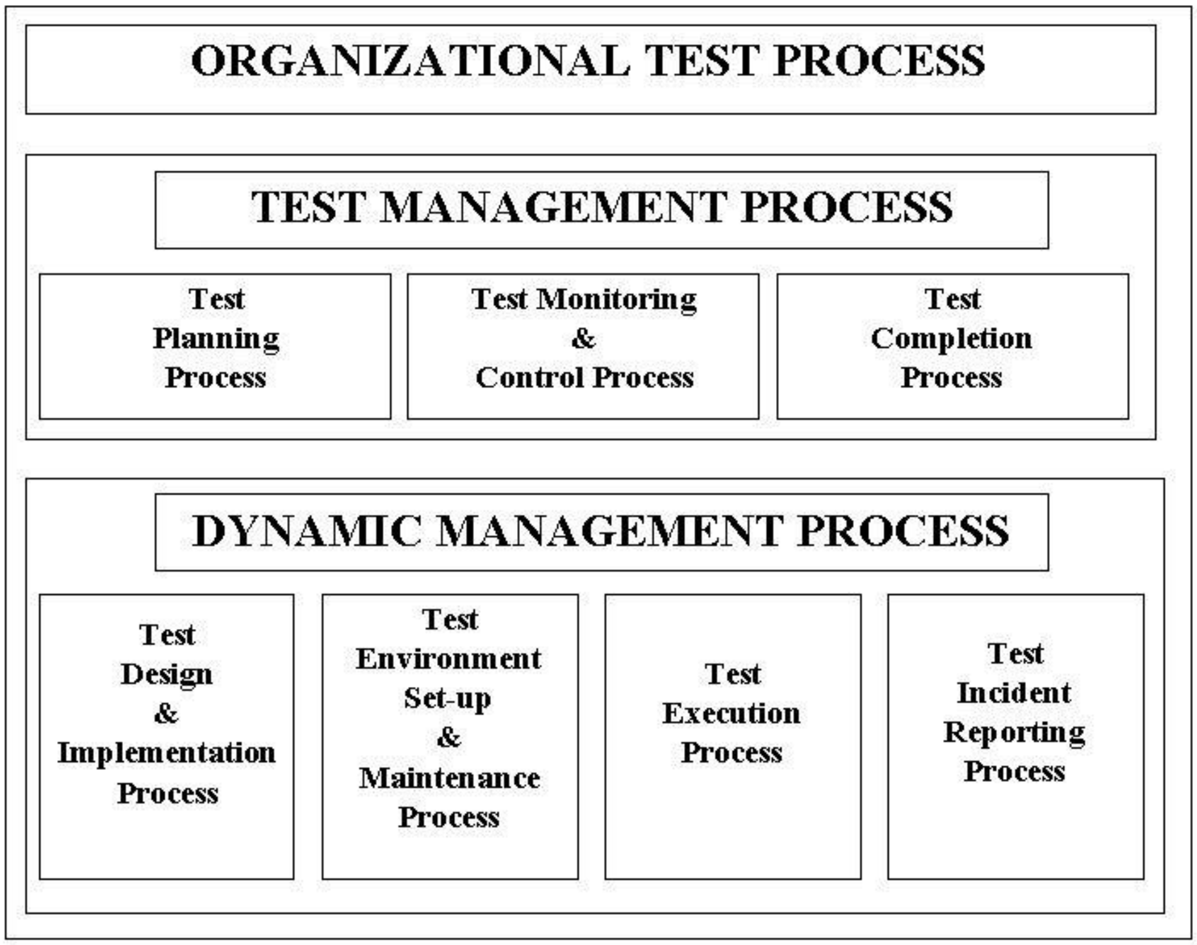

ORGANIZATIONAL TEST PROCESS
TEST MANAGEMENT PROCESS
Test Planning Process
Test Monitoring & Control Process
Test Completion Process
DYNAMIC MANAGEMENT PROCESS
Test Design & Implementation Process
Test Environment Set-up & Maintenance Process
Test Execution Process
Test Incident Reporting Process

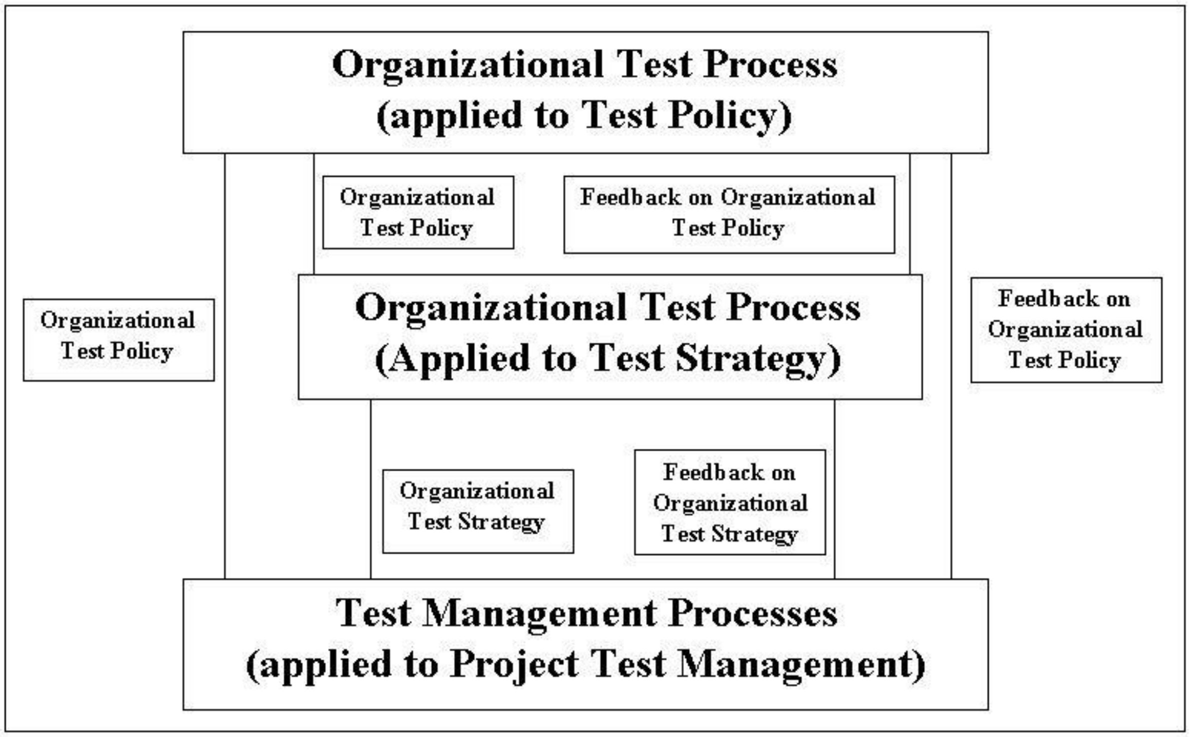
Figure 7.1: The multi-layer model showing all test processes

The “Organizational Test Policy” is a high-level document that specifies the organization's testing purpose, goals, and general scope. It also defines corporate testing processes and offers a framework for developing, assessing, and continuously improving its “Test Policy”, ”Test Strategy”, and project test management methodology.

The “Organizational Test Strategy” is a thorough, technical document which is outlines how testing is conducted throughout the business. It is a public document that gives recommendations for various initiatives within the company and is not project-specific.

The organizational test process is depicted in Figure 7.2 modified from [6]. Two instances of the organizational-level processes communicate with one another. The Organizational Test Strategy must be consistent with the Organizational Test Policy, and input from this activity should be presented to the Test Policy for future process improvement.

Similarly, the test management processes used on each organization's projects must align with the Organizational Test Strategy (and Policy). Feedback from project management is used to improve the organizational test process, which formulates and maintains the organizational test specifications.



Organizational Test Process (applied to Test Policy)
Organizational Test Policy
Feedback on Organizational Test Policy
Organizational Test Policy
Organizational Test Process (Applied to Test Strategy)
Feedback on Organizational Test Policy
Organizational Test Strategy
Feedback on Organizational Test Strategy
Test Management Processes (applied to Project Test Management)

Figure 7.2: Example Organizational Test Process implementation

The Organizational Test Process includes developing, reviewing, and maintaining organizational test standards as seen in Figure 7.3 modified from [6].

7.3 Test Management Processes

Test management consists of three processes; test planning, test monitoring and control, and test completion.

These general test management techniques may be used at the project level, at multiple test stages, and for handling various test types.

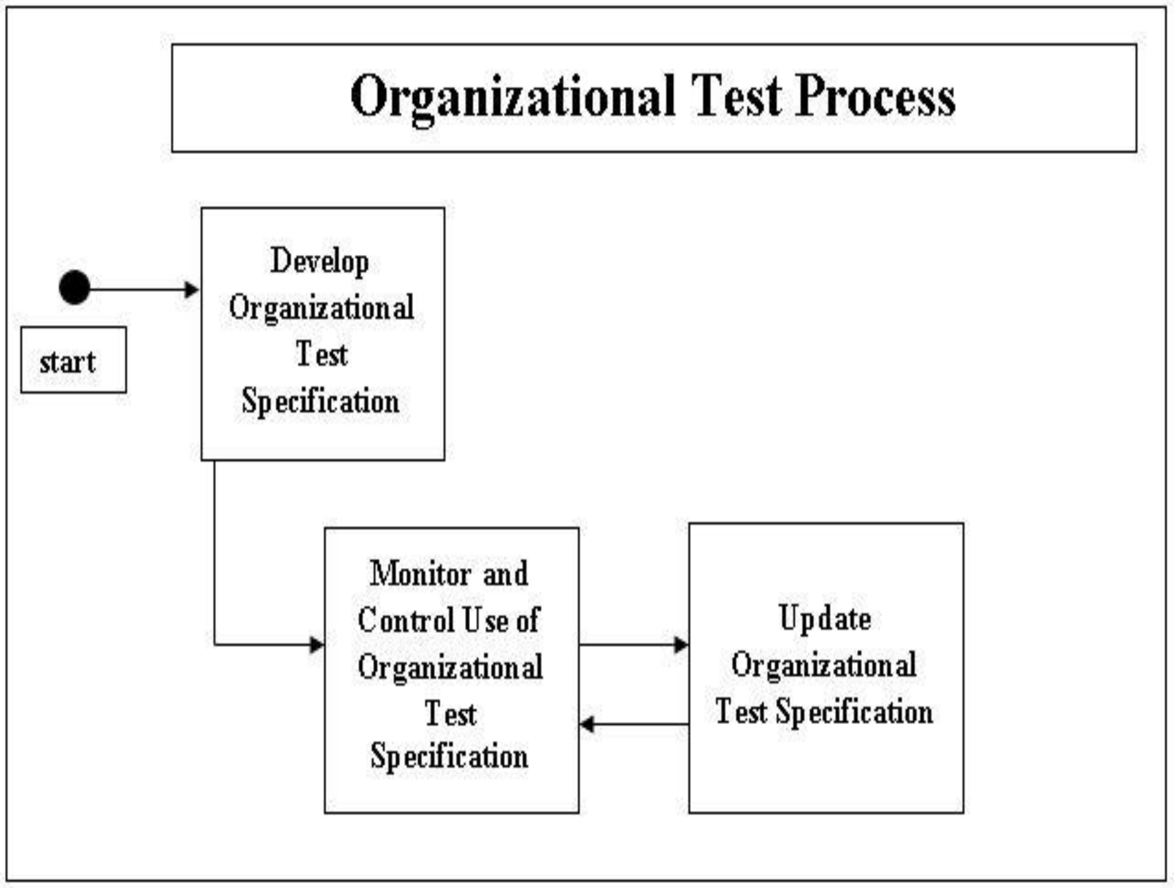

Organizational Test Process
start
Develop Organizational Test Specification
Monitor and Control Use of Organizational Test Specification
Update Organizational Test Specification

Figure 7.3: Organizational Test Process

When employed at the project test management level, these test management techniques are used to test the whole project using a project test plan.

Figure 7.4 modified from [6], depicts the links between the three test management processes and how they interact with the organizational test process, additional test management process applications, and dynamic test processes, which are thoroughly discussed in the following parts.

The organizational test process's outputs, such as the “Organizational Test Policy and Organizational Test Strategy”, must be aligned with the test management procedures.

In addition, the test management procedures may generate feedback on the organizational test process based on the practical execution of these results [6].

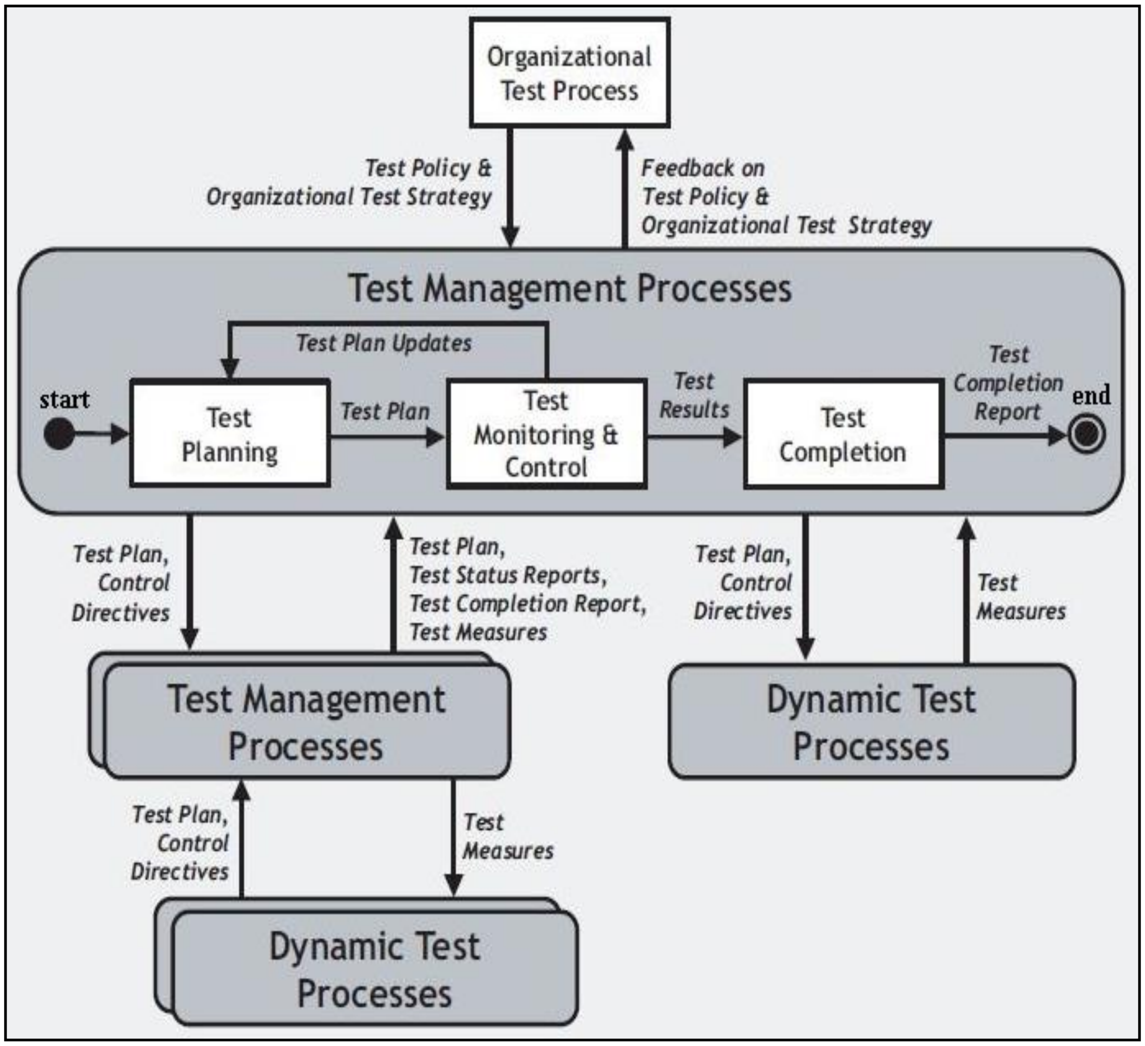

Organizational Test Process
Test Policy & Organizational Test Strategy
Feedback on Test Policy & Organizational Test Strategy
Test Management Processes
Test Plan Updates
start
Test Planning
Test Plan
Test Monitoring & Control
Test Results
Test Completion
Test Completion Report
end
Test Plan, Control Directives
Test Plan, Test Status Reports, Test Completion Report, Test Measures
Test Plan, Control Directives
Test Measures
Test Management Processes
Dynamic Test Processes
Test Plan, Control Directives
Test Measures
Dynamic Test Processes

Figure 7.4: Example test management process relationships [6]

7.4 Test Planning Process

In order to create the Test Plan, the Test Planning Process is employed. Depending on where this procedure is applied in the project, this might be a Project Test Plan. Test plan for a single-phase, such as a “System Test Plan”, or a test plan for a specific sort of testing, such as a “Performance Test Plan”.

The actions depicted in Figure 7.5 modified from [6], must be completed to build a test plan. As content for the test plan becomes available due to executing the stated actions, a draft test plan will be developed gradually until the entire test plan is documented.

Since the process is iterative, some of the tasks depicted In Test Planning Process Figure 7.5 may need to be revisited before the entire test plan is available and thoroughly discussed in the following sections.

7.5 Test Monitoring and Control Process

The “Test Monitoring and Control Process”, depicted in Figure 7.6 modified from [6], examines whether testing proceeds in line with the Test Plan. If there are considerable deviations from the intended progress, activities, or other parts of the test plan, activities will be undertaken to remedy or compensate for the resulting variances.

This technique may be used to oversee the testing of an overall test project or testing a single test type. In the latter situation, it is used as part of the “Dynamic Test Processes” monitoring and control of dynamic testing.

The “Test Monitoring and Control Process” aims to assess if testing is proceeding in line with the Test Plan and organizational test requirements.

It also executes control actions as needed and detects any adjustments to the Test Plan such as revise completion criteria or identify new actions to compensate for deviations from the Test Plan. The procedure is also used to verify whether testing is progressing with higher-level test plans, such as the Project Test Plan. Managing testing was done during specific test stages or for certain test types [6].

7.6 Test Completion Process

The Test Completion Process, depicted in Figure 7.7 modified from [6], is carried out once all testing activities have been agreed upon. It will be carried out to finish the testing carried out during a specific test phase or test type and complete the testing for the entire project.

The Test Completion Process's goal is to make usable test materials available for subsequent use, depart the test environment in a good state, and document and communicate the testing findings to key stakeholders [6].

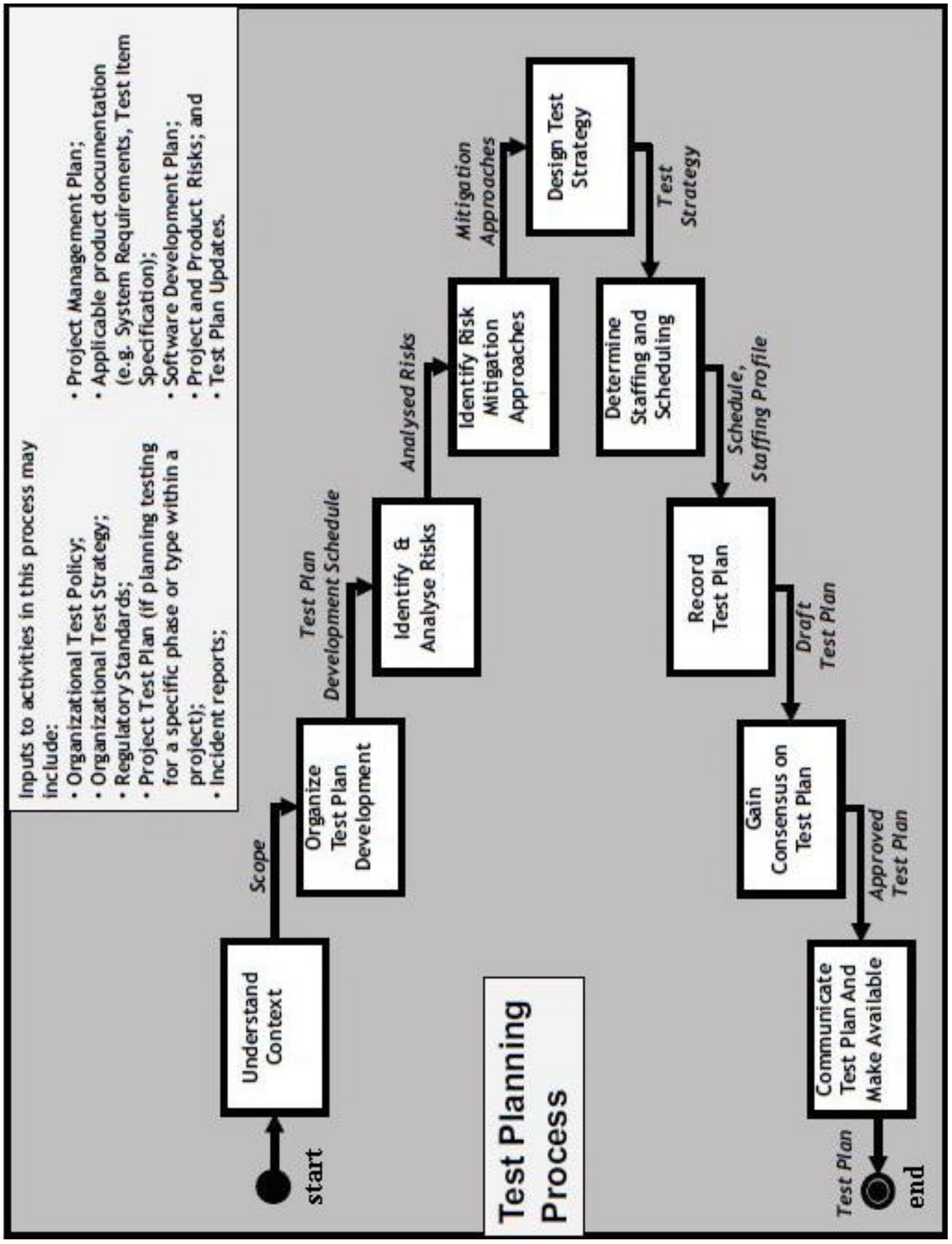
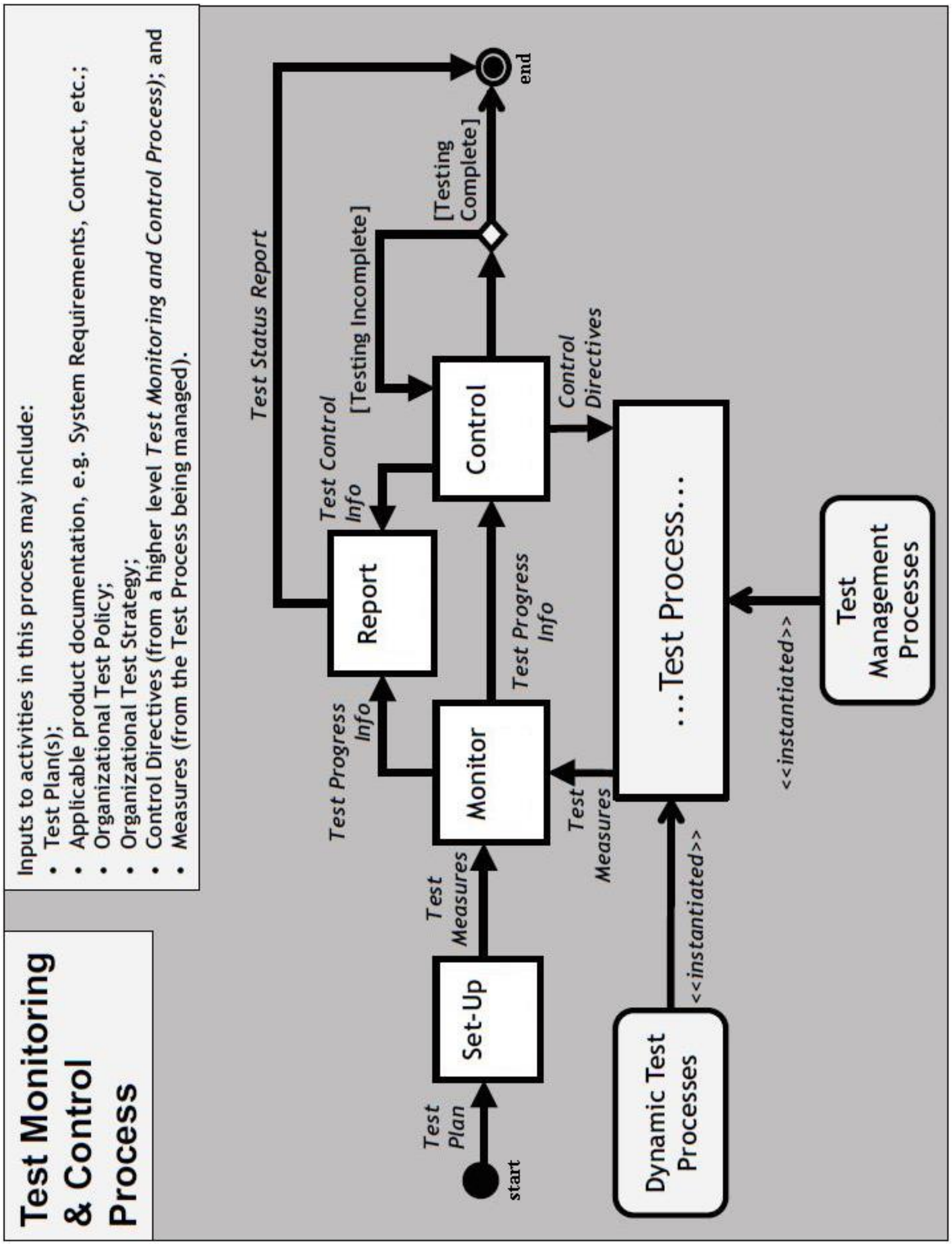

Inputs to activities in this process may include:
• Organizational Test Policy;
• Organizational Test Strategy;
• Regulatory Standards;
• Project Test Plan (if planning testing for a specific phase or type within a project);
• Incident reports;
• Project Management Plan;
• Applicable product documentation (e.g. System Requirements, Test Item Specification);
• Software Development Plan;
• Project and Product Risks; and
• Test Plan Updates.
start
Understand Context
Scope
Organize Test Plan Development
Test Plan Development Schedule
Identify & Analyse Risks
Analysed Risks
Identify Risk Mitigation Approaches
Mitigation Approaches
Design Test Strategy
Test Strategy
Determine Staffing and Scheduling
Schedule, Staffing Profile
Record Test Plan
Draft Test Plan
Gain Consensus on Test Plan
Approved Test Plan
Communicate Test Plan And Make Available
Test Plan
end
Test Planning Process

Figure 7.5: Test Planning Process [6]



Inputs to activities in this process may include:

- Test Plan(s);
- Applicable product documentation, e.g. System Requirements, Contract, etc.;
- Organizational Test Policy;
- Organizational Test Strategy;
- Control Directives (from a higher level *Test Monitoring and Control Process*); and
- Measures (from the Test Process being managed).

Figure 7.6: Test monitoring and control process [6]

7.7 Dynamic Test Processes

"Dynamic Test Processes" are used to do dynamic testing inside a specific testing phase. Four dynamic test procedures which are "Test Design and İmplementation", "Test Environment Setup and Maintenance", "Test Execution", and "Test Incident Reporting". Figure 7.8 modified from [6], depicts how the dynamic test processes interact and their link to the test management procedures. These dynamic test procedures are often triggered as part of the implementation of the test strategy defined in the test plan for the test phase such as system testing or test type such as performance testing that is being conducted [6].

The dynamic test processes will execute in the sequence depicted in "Dynamic Test Processes" shown in Figure 7.8 for every given test. However, these processes will often be triggered several times to complete a specific test phase or test type. Test measures, which are an output of the dynamic test processes and an input into the "Test Monitoring and Control Process" shown in Figure 7.6, can be generated during any dynamic test process activity. Test measurements are used to inform test management professionals about the state and progress of testing.

Test measurements, for example, might be used to inform test management about how many test cases the testing team has created.

On the other hand, control directives are an output of the "Test Management Process" and an input to the "Dynamic Test Process" Figure 7.6. They can be acted on during any activity of the dynamic test processes.

Control directives are instructions from test management personnel that specify how the test team should execute dynamic testing. For example, a control directive may be issued to a test team instructing them to build more test cases for new program features that their test manager has assigned to their team.

Test measures can be created during any activity of the dynamic test processes, and control directives can be acted upon during any activity of these processes [6].

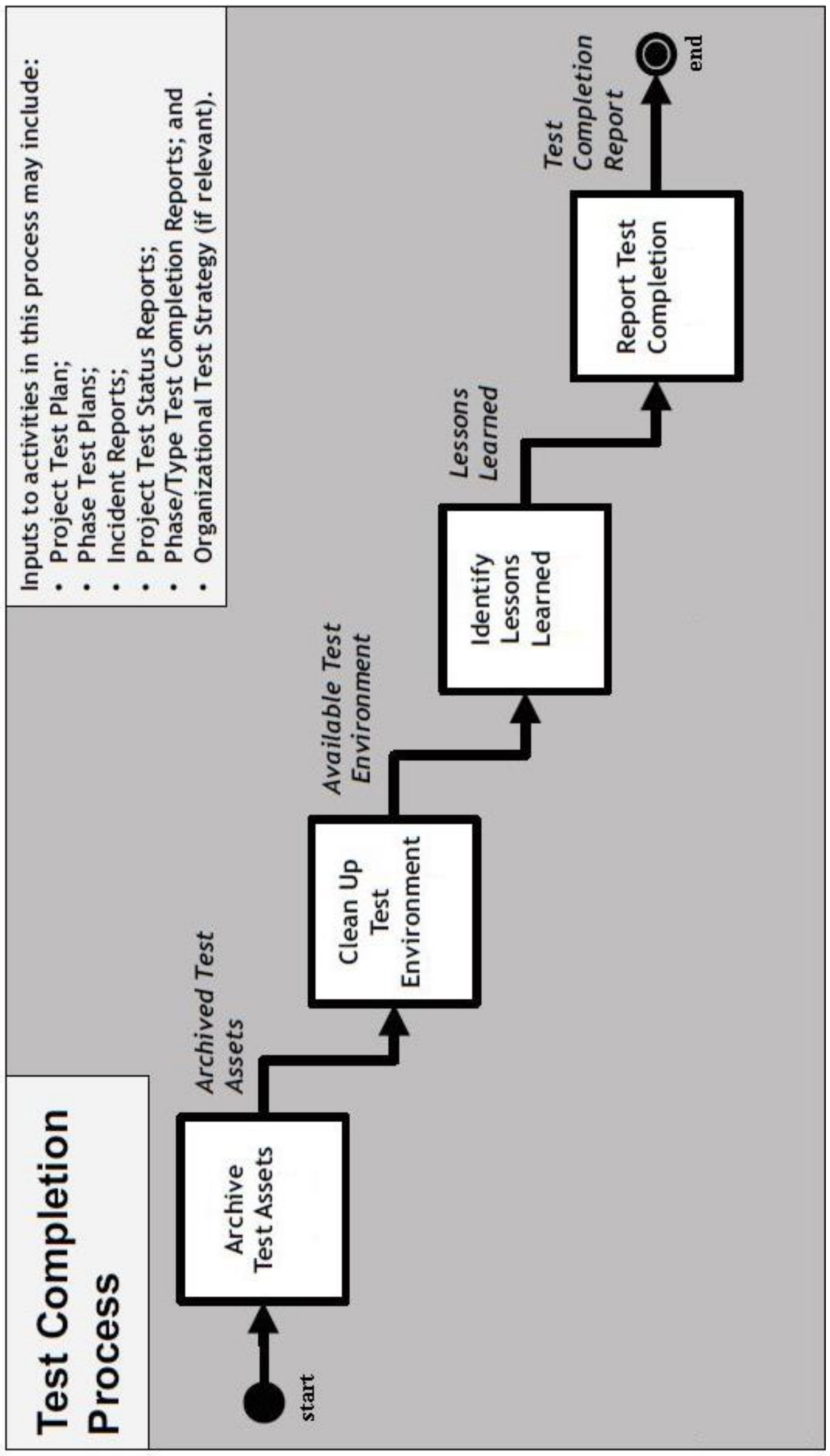

Test Completion Process
Inputs to activities in this process may include:
• Project Test Plan;
• Phase Test Plans;
• Incident Reports;
• Project Test Status Reports;
• Phase/Type Test Completion Reports; and
• Organizational Test Strategy (if relevant).
start
Archive Test Assets
Archived Test Assets
Clean Up Test Environment
Available Test Environment
Identify Lessons Learned
Lessons Learned
Report Test Completion
Test Completion Report
end

Figure 7.7: Test completion process [6]

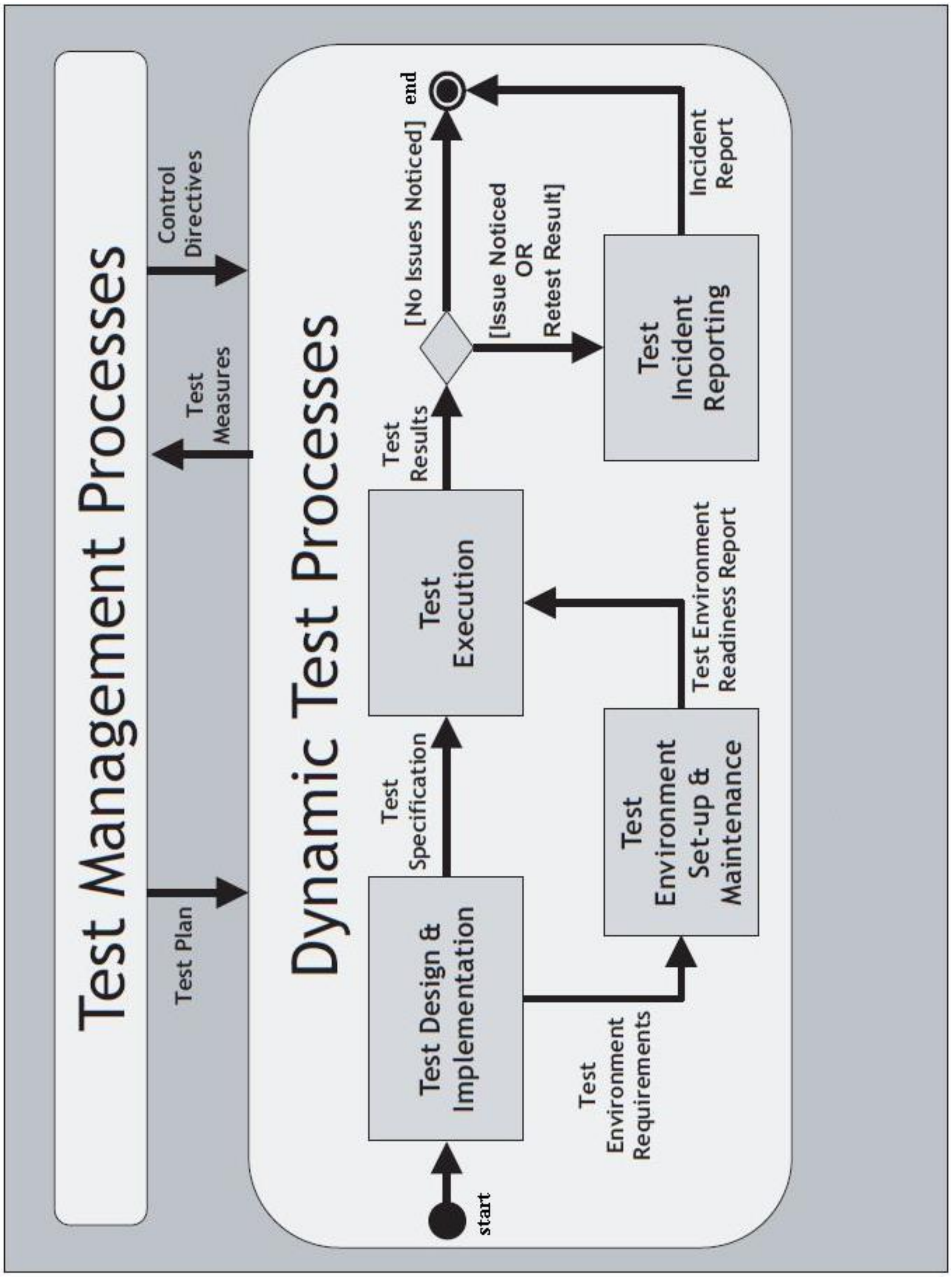

Test Management Processes
Test Plan
Test Measures
Control Directives
Dynamic Test Processes
start
Test Design & Implementation
Test Specification
Test Execution
Test Results
[No Issues Noticed]
end
[Issue Noticed OR Retest Result]
Test Incident Reporting
Incident Report
Test Environment Requirements
Test Environment Set-up & Maintenance
Test Environment Readiness Report

Figure 7.8: Dynamic test processes [6]

7.8 Test Design & Implementation Process

"The Test Design & Implementation Process" generates test cases and procedures, which are generally documented in a test specification. However, while doing exploratory testing, they are unlikely to be documented in advance. Therefore, the actions in Figure 7.9 modified from [6], are depicted in a logical sequence, although iteration will occur between many of them in actuality.

The Test Design & Implementation Process may also be quit and re-entered for various reasons. For example, it is discovered that additional test cases are necessary to fulfill the requisite test completion criteria after performing a test procedure or reporting an event.

This method necessitates testers to use one or more test design approaches to generate test cases and procedures with the ultimate goal of meeting the test completion requirements, which are often stated in test coverage metrics. The Test Plan specifies the test design approaches and also test completion criteria.

Various circumstances can cause iteration between actions in this process. For example, stakeholders may disagree with the outcome of an activity, such as identifying test conditions, in such cases. Similarly, scenarios may arise in which the outcomes of activity demonstrate that test planning decisions, such as selecting test completion criteria, are incompatible with project timing limitations, necessitating a reconsideration of "Test Management" practices.

The "Test Design & Implementation Process" is used to create test procedures that will be used throughout the "Test Execution Process". The test basis is examined, features are merged into feature sets, test conditions, test coverage items, test cases, test procedures are generated, and test sets are constructed as part of this process.

7.9 Test Environment Set-Up & Maintenance Process

"The Test Environment Set-Up & Maintenance Process" creates and manages the testing environment. The test environment's upkeep may necessitate alterations based on the outcomes of prior testing. Changes to test environments can be controlled using change and configuration management processes where they exist.

The "Test Plan" will first explain the needs for a test environment. However, the precise composition of the test environment will usually become evident once the "Test Design & Implementation Process" has begun.

The actions in Figure 7.10 modified from [6], are depicted as responsible for establishing and maintaining the needed test environment and communicating its status to all relevant stakeholders.

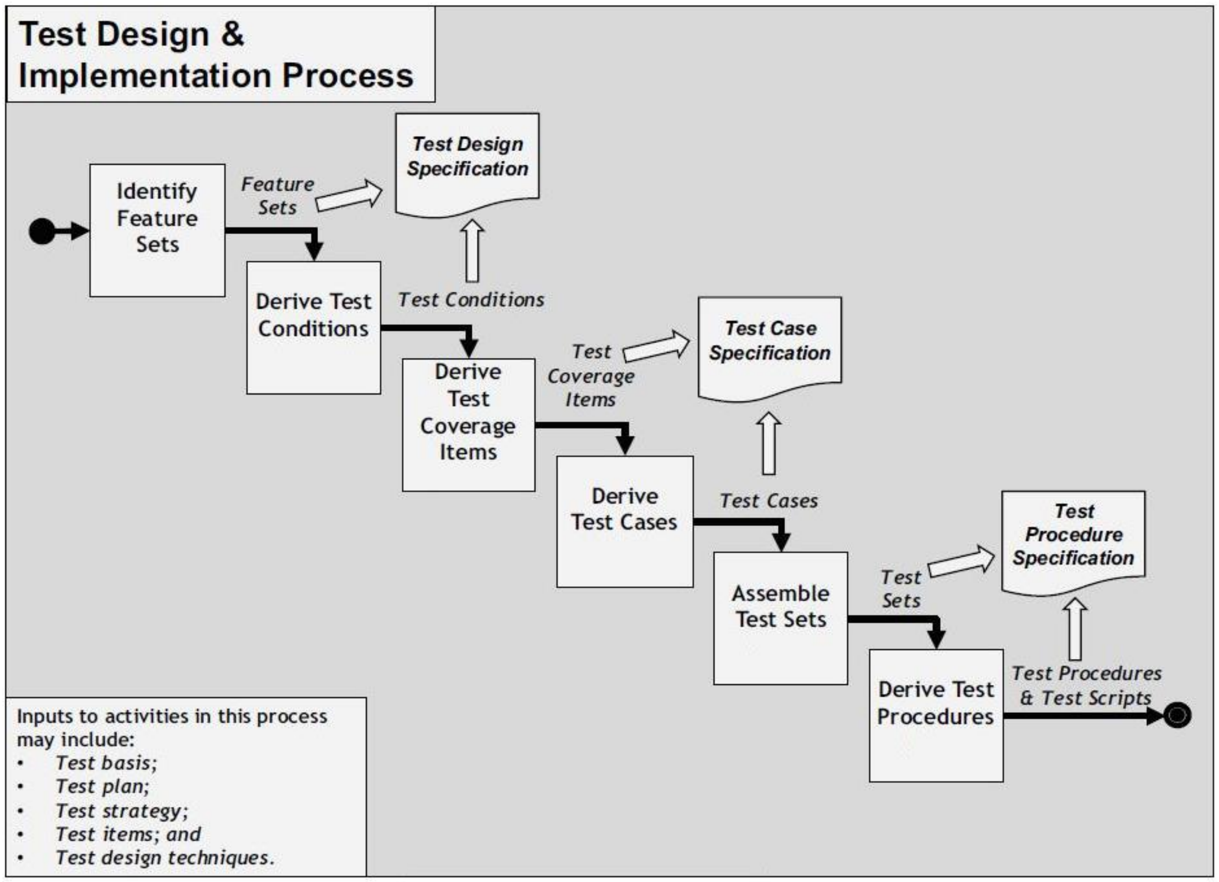


Figure 7.9: Test Design and Implementation Process [6]

7.10 Test Execution Process

The “Test Execution Process” is used to execute the test procedures created by the “Test Design & Implementation Process” on the test environment created by the “Test Environment Set-Up & Maintenance Process”.

All accessible test procedures cannot be conducted in a single cycle, and the “Test Execution Process” may need to be repeated. Therefore, if a problem has been resolved, it should be retested by restarting the Test Execution Process.

Figure 7.11 modified from [6], depicts the tasks logically, although repetition will occur between many of them in actuality. Typically, the comparison of test findings and the recording of test execution data will occur concurrently with the execution of test processes. The Test Execution Process's goal is to run the test procedures developed during the "Test Design & Implementation Process" in the prepared test environment and report the findings.

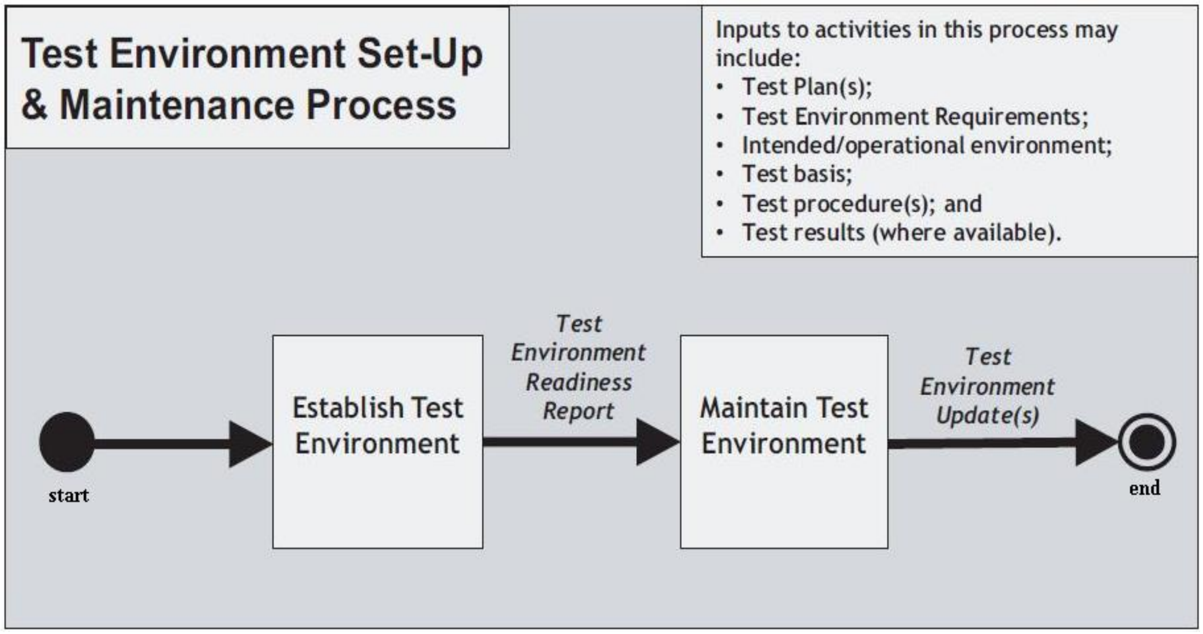

Test Environment Set-Up & Maintenance Process
Inputs to activities in this process may include:
• Test Plan(s);
• Test Environment Requirements;
• Intended/operational environment;
• Test basis;
• Test procedure(s); and
• Test results (where available).
start
Establish Test Environment
Test Environment Readiness Report
Maintain Test Environment
Test Environment Update(s)
end

Figure 7.10: Test Environment Set-Up & Maintenance process [6]

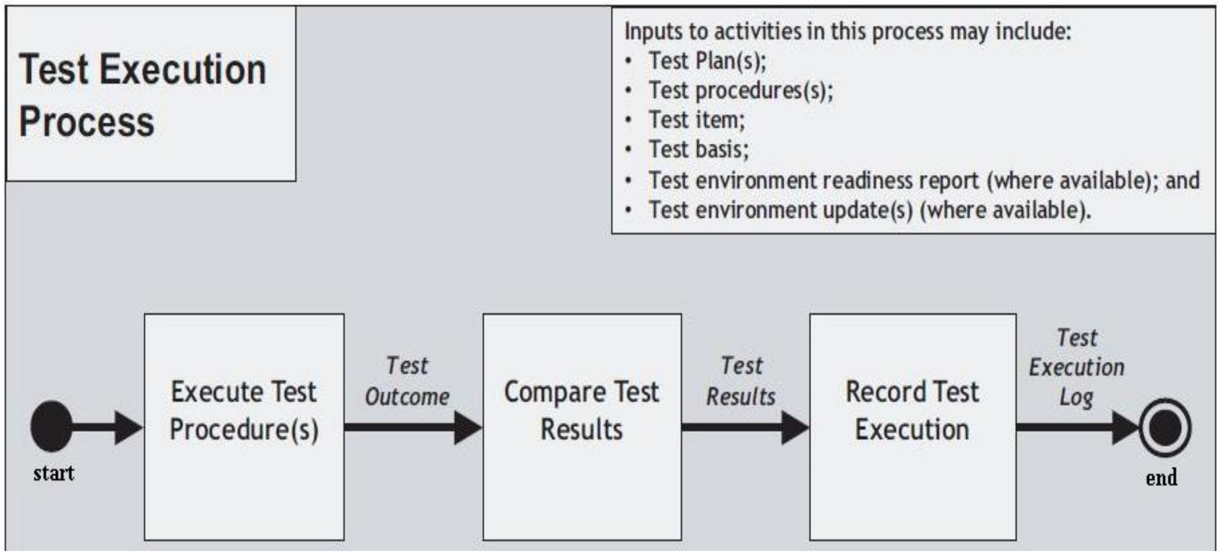

Test Execution Process
Inputs to activities in this process may include:
• Test Plan(s);
• Test procedures(s);
• Test item;
• Test basis;
• Test environment readiness report (where available); and
• Test environment update(s) (where available).
start
Execute Test Procedure(s)
Test Outcome
Compare Test Results
Test Results
Record Test Execution
Test Execution Log
end

Figure 7.11: Test Execution process [6]

7.11 Test Incident Reporting Process

Test incidents are reported using the Test Incident Reporting Process. This procedure will be initiated due to test failures, occasions when something strange or unexpected occurred during test execution, or when a retest succeeds.

The goal of the Test Incident Reporting Process is to notify key stakeholders of occurrences that require additional action as a result of test execution. Figure 7.12 modified from [6], shows that in the case of a new test, this will necessitate the creation of an incident report. Likewise, in the case of a retest, the status of a previously-raised incident report must be updated. However, it may also necessitate the creation of a new incident report if additional occurrences are discovered.

7.12 Static Testing

"Static Testing" relies on manual review of work items or tooled assessment of code or other work items [3], [4].

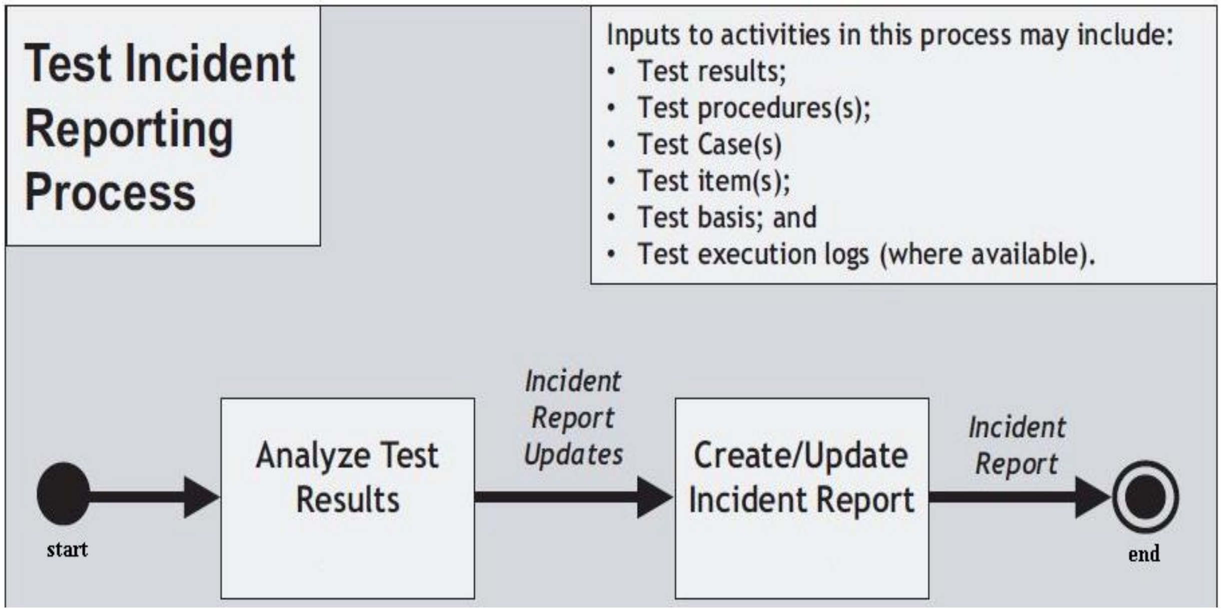

Test Incident Reporting Process
Inputs to activities in this process may include:
• Test results;
• Test procedures(s);
• Test Case(s)
• Test item(s);
• Test basis; and
• Test execution logs (where available).
start
Analyze Test Results
Incident Report Updates
Create/Update Incident Report
Incident Report
end

Figure 7.12: Test Incident Reporting process [6]

Advantages of "Static Testing";

- Detecting and fixing failures more rapidly and before sophisticated tests are run;

- To avoid specification or coding problems, detect inconsistencies, ambiguities, anomalies, omissions, inaccuracies, and redundancies. Increasing the production efficiency;
- Researching at a lower cost and for a shorter period;
- Due to fewer faults later in the lifecycle, they lower the average cost of quality during the software's lifespan.

7.13 Differences between Static & Dynamic Testing

"Static Testing" finds flaws in work products directly. The detection errors caused by bugs after the software has been executed. Consequently, static testing able to detect the fault with far less effort.

As a result, static testing can increase work product consistency and internal quality, whereas dynamic testing focuses on externally observable behaviors [3], [4].

The following are examples of common problems that are easier and less expensive to detect and rectify using static tests than with Dynamic Testing:

- Requirement errors;
- Design errors;
- Coding errors;
- Deviations from standards;
- Incorrect interface requirements;
- Vulnerabilities;

The items are listed above show essential features of the static and the dynamic test methods. It is seen that static and dynamic tests are carried out together in most projects, and they are also handled in separate phases depending on the project context.

7.14 Review Processes

Informal to formal reviews are possible. However, team participation, recorded review results, and documented review techniques distinguish formal reviews.

The planned aims of the review decide the review's topic.

7.15 Work Product Review Process

The following are the primary actions that contribute to the review process: [3], [4];

- Planning;
- Initiate review;
- Individual review;
- Communication and analysis of the issues;
- Fixing errors and reporting.

7.15.1 Planning

- Defining review scope;
- Determining review purpose;
- Estimate effort and time;
- Identifying review features such as review type, activities, and checklists;
- Selection of review participants and role sharing;
- Deciding entry and exit criteria for more formal kinds of evaluations;
- Checking that entry criteria are met.

7.15.2 Initiate review

- Dissemination of other materials such as finding checklists which are related to work products;
- Clarification of the scope, goals, process, roles, and work products;
- Answering the questions of the participants about the review.

7.15.3 Individual review

- Reviewing all parts of the work product;
- Note potential errors, findings, suggestions, and questions.

7.15.4 Communication and analysis of the issues

- Communication of possible errors identified;
- Analyzing potential errors, assigning a responsible and case for them;

- Evaluation and documentation of quality characteristics;
- Making a review decision;
- Evaluation of review findings according to exit criteria.

7.15.5 Fixing errors and reporting

- Creating error reports for findings requiring change;
- Resolving errors detected in the work product under;
- The appropriate person or forwarding to the team;
- With the consent of the commenter where possible, updated error conditions;
- Checking that exit criteria are met;
- The acceptance of the work product should meet the requirements for exit criteria.

7.16 Roles & Responsibilities in a Formal Review

Formal review includes the following roles:

- Author;
- Administration;
- Moderator;
- Review Leader;
- Reviewers;
- Scriber.

These roles and responsibilities in a formal review are described as follows;

7.16.1 Author

The author produces product under review and corrections errors in the work product under evaluation.

7.16.2 Administration

The administration responsible for review planning. Administrators have to decide on the implementation of reviews such as allocates staff, budget, and time.

7.16.3 Moderator

Moderator ensures that review meetings are conducted effectively.

7.16.4 Review leader

The review leader is responsible for the review, determines who will be engaged in the process, and plans when and where it will occur.

7.16.5 Reviewers

Reviewers’ experts on the subject, people working on the project, stakeholders having a say on the work product, and specific technical or people with work experience. Find findings in the work product, and reviewers can represent different perspectives such as tester, programmer, user, operator, business analyst, usability specialist.

7.16.6 Scriber

Scriber brings together and organizes the findings found during each review activity. Records potential errors and decisions identified at the review meeting.

7.17 Review Types

Reviews primary purposes is finding failures. Therefore, reviews can be classified various characteristics.

The most common types of reviews are listed below and their characteristics [3], [4].

- Informal review;
- Walkthrough;
- Technical review;
- Inspection.

These review types are defined below in more detail [3], [4]:

7.17.1 Informal review

- Main goal: finding potential bugs;
- A review meeting may not be held;

- It can be performed by the author's colleague (colleague check) or more;
- Results can be documented;
- The utility varies depending on the reviewers;
- The use of checklists is optional;
- Widely used in agile software development.

7.17.2 Walkthrough

- Main objectives: finding bugs, improving software, considering alternative applications, standards, and evaluation of compliance with the requirements;
- The author of the work product usually leads the review meeting;
- The printer is mandatory;
- The use of checklists is optional;
- It can be in the form of scenarios, rehearsals, or simulations;
- Potential error records and review reports can be generated;

7.17.3 Technical review

- Reaching consensus, and finding potential bugs;
- Other purposes: to evaluate the quality and to build trust for the work product, to generate new ideas;
- Reviewers selected from technical experts in other or the same disciplines who are technically equivalent to the author;
- Individual preparation is required prior to the review meeting.

7.17.4 Inspection

- Main objectives: Finding potential defects, assessing the quality of the work product and building trust, learning by the authors of the work product and preventing similar mistakes in the future through root cause analysis;
- Individual preparation is required prior to the review meeting;
- Reviewers are the equivalent of the work product author or experts in other disciplines related to the work product;

- Determining entry and exit criteria are used;
- There must be a scriber;
- Writer; the review leader cannot act as a reader or writer;
- Potential error records and review reports are created.

A single work product can be subjected to many types of reviews. For example, an informal review may be undertaken prior to the “Technical Review” confirm that the working product is ready for “Technical Review”. Furthermore, evaluations mentioned above might be carried out as peer reviews by colleagues at a comparable organizational level. The mistakes discovered throughout the review process differ, primarily due to the work product under evaluation.

7.18 Applying Review Techniques

During the individual review (individual preparation) activity, various review strategies can be used to discover mistakes. These strategies can be employed in any of the reviews mentioned above kinds. Depending on the sort of review conducted, the efficiency of approaches may differ. Categories are mentioned below as examples:

- Ad-hoc;
- Checklist-based;
- Scenarios and dry runs;
- Perspective-based;
- Role-based.

These review techniques are defined below in more detail [3], [4]:

7.18.1 Ad-hoc

There is little or no direction for reviewers on doing this duty in the ad-hoc review. Reviewers often read the work output in order and keep track of any issues. The approach of non-editing review is commonly utilized and needs little preparation. This method is mainly reliant on the reviewers' abilities, and it can result in a slew of similar issues being identified by many reviewers.

7.18.2 Checklist-based

This method in which reviewers identify flaws using checklists provided at the outset of the process (for example, by the moderator). The review checklist is made up of questions that have been generated from experience and are based on likely mistakes.

The most significant benefit of this approach is the systematic covering of common types of errors. Individual evaluations should be done with caution, rather than just following the checklist and looking for problems outside of it.

7.18.3 Scenarios and dry runs

Guidelines on reading the work product are provided to reviewers in the scenario-based review. The scenario-based approach (if the work product is documented correctly, such as usage scenarios) facilitates the reviewers' job by enabling "rehearsals" on the work product. These scenarios provide better guidance to reviewers in finding errors than simple checklists. Reviewers should not be limited to documented scenarios to avoid missing other types of errors such as missing features, checklist-based reviews.

7.18.4 Perspective-based

During the individual review phase of perspective-based reading, reviewers evaluate diverse stakeholder viewpoints, similar to role-based reviews. For example, End-user, marketing, designer, tester, and enterprise management are standard stakeholder views.

Using multiple stakeholder viewpoints adds richness to individual reviews and minimizes the possibility of reviewers discovering similar concerns.

Reviewers must also strive to use the work result when doing perspective reading. For example, suppose a tester reads the requirements from the end-users perspective.

In that case, he will try to write draft acceptance tests to determine if all of the necessary information is provided. In addition, checklists are anticipated to be utilized for perspective-based readings.

According to studies, the most successful approach for reviewing requirements and technical study products is perspective-based reading.

7.18.5 Role-based

Role-based review is a technique where reviewers evaluate the work product from the perspective of the roles of each stakeholder affected by the work product.

7.19 Test Activities & Tasks

Software testing is a set of steps to place specific test-case design techniques, and testing methods should be defined for the software process. In the literature, a variety of software testing techniques have been suggested. Different testing strategies are acceptable for various software engineering methods and at different points in time. Testing is run by the developer of the software and independent test groups.

A test process includes following activities:

- Test planning;
- Test monitoring and control;
- Test analysis;
- Test design;
- Test implementation;
- Test execution;
- Test completion.

Activities that are listed in the subsections below, make up each essential category of activities. Furthermore, while all of these key task classes tend to be introduced in a logical order, they are often implemented iteratively. Agile architecture, for example, entails minor revisions of program design, development, and testing that occur consistently and are accompanied by ongoing planning.

As a result, research tasks are also carried out under this software development strategy on an iterative, continuous basis. Even in sequential software development, there will be duplication, mix, concurrency, or exclusion in the logical sequence of leading activity groups [3], [4].

This group of activities is defined below in more detail [3], [4]:

7.19.1 Test planning

Test preparation entails actions that identify the research goals and the strategy for achieving those objectives within the context's constraints. Then, test plans can be revised based on input from testing and control operations.

7.19.2 Test monitoring and control

Test monitoring entails continuously comparing real progress to expect. The assessment of exit requirements, also known as the concept of software development lifecycle models, aids test tracking and regulation. In test progress updates, stakeholders are informed about progress against the schedule, including variations from the plan and facts to justify any decision to stop testing.

7.19.3 Test analysis

The test basis is examined during test analysis to distinguish testable characteristics and related test conditions. Using black-box, white-box, and experience-based evaluation methods will help minimize the risk of omitting critical test conditions and determine more detailed and reliable test conditions during the test analysis process.

Test analysis may also result in test scenarios used as research goals in test charters. Coverage obtained during experience-based training will be calculated as these research objectives can be traced to the test basis.

7.19.4 Test design

During the test design process, the test conditions are elaborated into high-level test cases, collections of high-level test cases, and other test ware. As a result, research design asks "how to test?" and test analysis answers the question "what to test?"

Main events are included in the test design;

Creating, prioritizing, and designing test cases and collections of them locating test data used to help test conditions and test cases.

Creating test environments and determining any facilities and resources between the test basis, test environments, and test cases are captured.

7.19.5 Test implementation

The test ware required for test execution is developed and completed during test implementation, including sequencing test cases into test procedures.

The functions of test design and test execution are often intertwined.

7.19.6 Test execution

Test suites are performed in conjunction with the test execution schedule during test execution. Keeping track of the test items or test object's numbers, as well as the test tools and test ware, manually or with the aid of software execution equipment, run experiments. First, actual and predicted outcomes are compared. Then, they analyze irregularities to determine their possible causes (for example, errors can occur due to code flaws, but false positives can also occur).

They are reporting faults based on reported errors. In addition, they are keeping track of test execution results (e.g., pass, fail, blocked). Bi-directional traceability between the test basis, test scenarios, test events, test protocols, and test outcomes are checked and updated.

7.19.7 Test completion

Test completion tasks occur when a development framework is launched, a prototype product is completed, an agile product version is completed, a test stage is completed, or a fixing is completed.

8. TEST TECHNIQUES

8.1 General

Test approaches are listed as “Black-box”, “White-box”, or “Experience-based” in this chapter. “Black-box Testing” methodologies are based on an assessment of the appropriate test foundation, according to the ISTQB [3], [4].

Both functional and non-functional testing may be done with these approaches. Inputs and outputs of the test items are the focus of black-box testing, which ignores the test object's underlying structure.

White-box testing (also known as structural or structure-based testing) examines the architecture, precise design, internal structure, or coding of the test item.

In contrast to black-box testing methodologies, “White-box Testing” focuses on structure and operations of test object. “Experience-based Testing” approaches generate, construct, and execute tests based on developers', testers,' and users' knowledge and experience. These methods are typically used with black-box and white-box testing strategies.

8.2 Test Types

A test activities analyzes specific aspects of a software system or a system component concerning testing objectives.

Completeness, correctness, and appropriateness are all functional quality elements that are assessed.

Non-functional quality factors are assessed, such as reliability, performance, security, and compatibility [3], [4].

The software and system product quality process is depicted in Figure 8.1 modified from [8], and Figure 8.2 modified from [8].

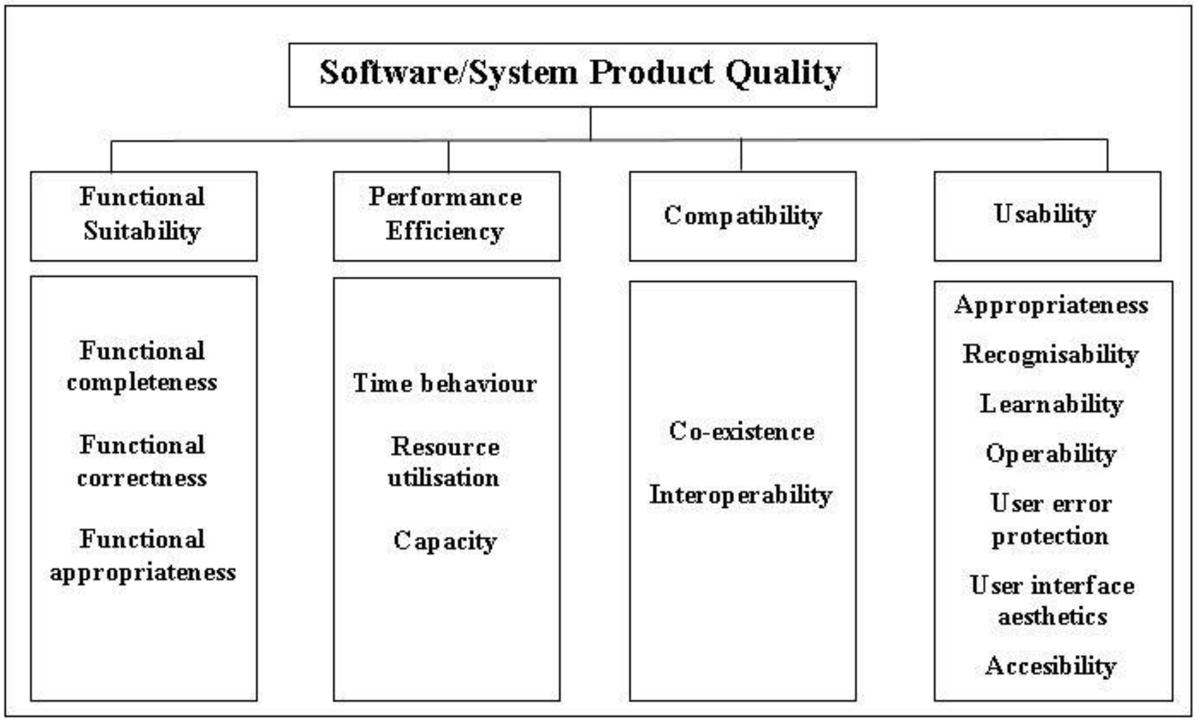


Figure 8.1: Product quality model

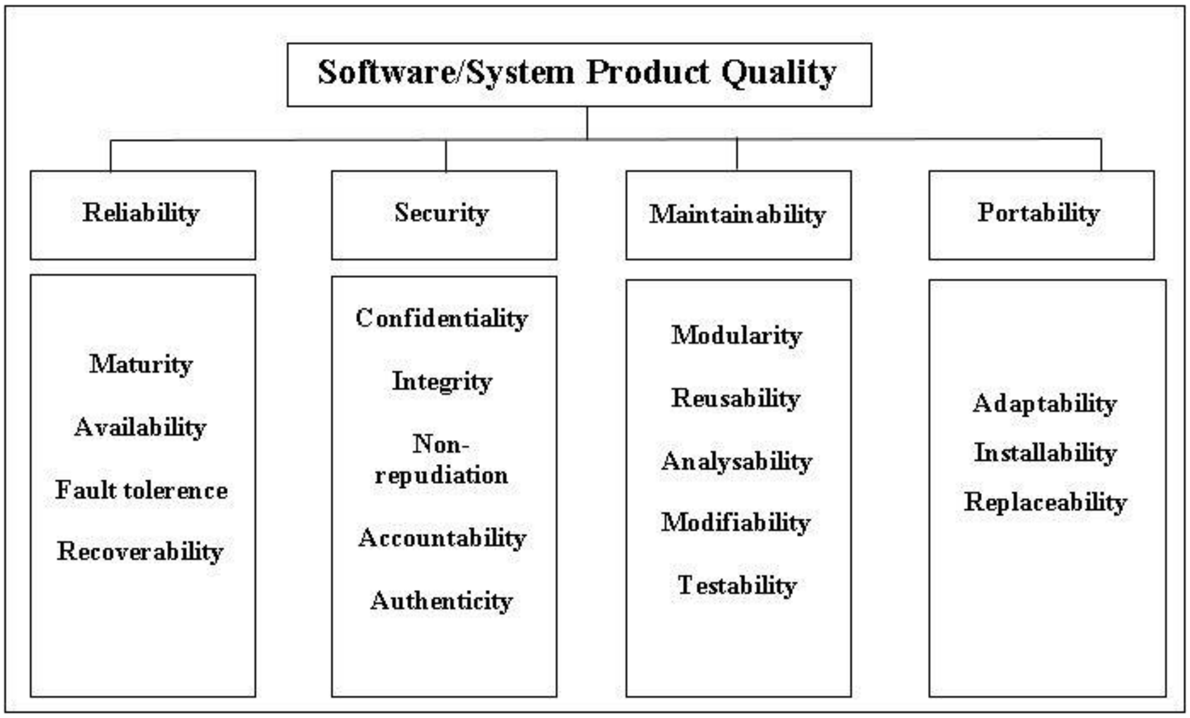


Figure 8.2: Product quality model (cont'd)

8.3 Functional Testing

"Functional Testing" which is related to system comprises tests' that evaluate the system's functions. At all levels of testing, functional tests should be performed. The focus, however, differs. Functional coverage is the percentage of a component covered in which tests have exercised some functionality [3], [4].

8.4 Non-functional Testing

System and software attributes such as usability, performance efficiency, and security are assessed during non-functional testing. Non-functional Testing evaluates the system's ability.

The level to which tests have exercised some non-functional element is known as non-functional coverage [13], [3], [4].

8.5 Change-related Testing

Some changes are added in a system, whether to cure a defect or add or update functionality, testing should ensure that the modifications have appropriately addressed the problem or implemented the feature with no unintended repercussions [3], [4].

8.5.1 Confirmation testing

"Confirmation Testing is continued when failure is fixed. The program also be subjected to new tests to cover and required to correct the fault.

The actions to replicate the failures caused by the fault must be re-executed on the updated software version at the very least [3], [4].

8.5.2 Regression testing

In the context of an integration test approach, regression testing is the re-execution of a subset of previously performed tests to check that modifications have not resulted in unforeseen side effects. Regression testing ensures that modifications do not create unexpected behavior or problems. Manual regression testing may be performed by re-running a subset of all test cases or utilizing automated capture/playback tools [3], [4].

8.6 Black-box Test Techniques

Black box tests are not concerned with the code's structure, design, and implementation. Instead, the system's operation is tested according to the input and output changes in black-box tests. Most software testers commonly use black-box test variants. According to ISTQB, black-box testing, also known as behavioral or functional testing, focuses on the software's functional specifications [3], [4]. In other words, black-box testing strategies allow one to create collections of input conditions that entirely exercise all of a program's functional specifications. However, Black-box testing is not a replacement for white-box testing. Instead, it is a supplement to white-box approaches, and it is more apt to discover a different set of mistakes.

8.6.1 Equivalence partitioning

“Equivalence Partitioning” is a kind of Black-box Testing” method that partitions a program's input data classes from which test cases can be produced. An equivalence class exists if a set of objects can be connected by symmetric, transitive, and reflexive relationships. An equivalence class shows a set of valid or invalid states for input conditions. A specific value is required for an input state, and one valid and two invalid equivalence classes are defined. An input condition specifies a set member, and one valid and one invalid equivalence class is defined.

Test cases for each input domain data object may be built and implemented using the criteria for equivalence class derivation. The test cases are chosen to simultaneously exercise the most significant number of properties of an equivalence class [14]. Equivalence partitioning splits data into partitions to expect each partition's members to be processed in the same manner.

The term "invalid equivalence partition" refers to an equivalence partition that contains invalid values. Each data variable connected to the test object. In order to ensure that errors are not masked, invalid equivalence partitions should be checked independently rather than in combination with other invalid equivalence partitions.

Coverage is expressed as a percentage of the total number of known equivalence partitions divided by the number of equivalence partitions evaluated by at least one value [3], [4].

Here is the sample of equivalence partitioning [15];

Only numeric characters are permitted in a text field. The length must be between 6 and 10 characters;

When analyzing “Equivalence Partitioning”, all values in all categories are equal, which is why 0-5 are equivalent, 6–10 are equivalent, and 11–14 are equivalent.

When testing, consider 4 and 12 to be incorrect numbers and 7 to be legitimate.

It is simple to test input ranges 6–10 but more difficult to test input ranges 2-600. Testing will be simple in fewer test cases, but one must use extreme caution. For example, assume that the valid input is 7. One believes the developer coded the proper valid range (6-10).

8.6.2 Boundary value analysis

“Boundary Value Analysis” is another “Black-box Testing” method that supports the equivalence classification technique. It is also called Boundary Test or Boundary Value Test. It is used to test numeric ordinal values' lower and upper bounds. When best practices are examined, it is seen that there is a density of errors in the processing of limit values in numerical inputs. For this reason, it is aimed to show that the Boundary Value (BV), one below the Boundary Value (BV-1), and one above the Boundary Value (BV+1) works as expected in terms of finding these errors in the limit value tests [16].

Here is the sample of boundary value analysis [15];

Test cases for an application with an input box that takes integers ranging from 1 to 1000. The valid range is 1 to 1000, the invalid range is 0, and the invalid range is 1001 or more.

Create test cases for the valid partition value, the invalid partition value, and the precise boundary value;

- Test Case 1: Assume the test data to be the input domain's input boundaries, i.e., values 1 and 1000.
- Test Case 2: Take test data with values between 0 and 999, which are close below the extreme boundaries of input domains.
- Test Case 3: Assume test data with values close above the extreme boundaries of the input domain, such as 2 and 1001.

8.6.3 Decision table testing

"Decision Table Testing" is a type of "Black-box Test" design approach used to examine system behavior for various input combinations. Decision tables are an excellent approach to capture the complicated business rules that a system must follow. It is a methodical methodology in which various combinations of inputs and their related system behavior, outputs which are described in tabular form.

In requirement specifications, input-output combinations can also be represented as cause-and-effect graphs. As a result, in the literature, decision table tests are called cause-effect graph tests. Decision Tables are tabular representations of inputs concerning rules, cases, and test conditions [16].

8.6.4 State transition testing

Based on current conditions or experience, components or systems can react to an event differently. The definition of states can be used to summarize the experience. The program can take action as a result of the state change. A state transition table lists both legitimate and theoretically invalid state transitions and the events and behavior that occur from valid transitions.

State transfer monitoring is commonly used in the embedded computing industry for menu-based applications. The method can also be used to model a business situation of several states or to analyze screen navigation [3], [4].

8.6.5 Use case testing

This testing interactions between events, as well as preconditions, post-conditions, it will also be used to define a use case. The subject's state can change due to interactions between the actors and the subject. Workflows, activity diagrams, and business process models can also be used to visualize interactions [3], [4].

8.7 White-box Test Techniques

In this testing, the accuracy and quality of the code are tested by entering the code. Therefore, code access is required for this test type. Tests for code structure and design are carried out. For example, an unnecessary block of code can be detected, or situations aimed at increasing the readability of the code can be detected. Errors to be found early in the code also facilitate “Black-box Testing”.

“White-box Testing” is mainly done by developers as well as by testers. According to ISTQB, “White-box Testing” is based on the test object's internal structure [3], [4]. “White-box Testing” approaches may be used at all levels of testing.

8.7.1 Statement coverage testing

It is assumed that by running the code in the expression scope, each statement in the code will be executed at least once. This ensures that none of the statements have any unintended consequences. Statement coverage is a type of “White-box Testing” design approach. A software's source code reveals that it has a wide range of features such as variables, constants, functions, loops, controllers, and error catchers. Some of these statements will be performed, while others will be left unexecuted, based on the test scenarios submitted by the testers. The goal of statement coverage is to demonstrate how many of all possible statements in the code have been run with the test cases executed [16].

The following formula is used to calculate statement coverage:

Statement Coverage = (Number of Statement Run / Total Statement in Code) * 100

8.7.2 Decision coverage testing

Decision coverage is also known as branch coverage. When the software codes are examined, it is seen that no code progresses in a static structure. Instead, there are decision points in the code.

The software flow is divided into alternative branches at these decision points. Decision coverage aims to show that these decision points in the code show the expected behavior and that each branch is run at least once.

Decision points correspond to IF statements in the code. An IF statement is divided into true and false branches [16].

The following formula is used to calculate decision coverage:

Decision Coverage = (Number of Decision Results / Total Code Decision Results) * 100

8.7.3 The importance of statement and decision testing

When one hundred percent statement coverage is achieved, all executable statements in the code have been tested at least once, but not all decision mechanisms have been tested. As a result, one of the two “White-box Methodologies”, statement testing, may provide less coverage than decision testing.

When it reaches one hundred percent decision coverage, it tests all choice outcomes, including accurate and false outcomes, even though no explicit false statement is present.

Statement coverage assists in the discovery of flaws in programs that have not been subjected to another testing.

If previous tests have not considered both true and false outcomes, decision coverage can help find flaws in code. Obtaining one hundred percent decision coverage assures one hundred percent statement coverage (but not the other way around) [3], [4].

8.8 Experience-based Test Techniques

“Experience-based Testing” is not a traditional method, but it is still a powerful way to test an application. It is a hands-on approach where testers participate in minimum planning and maximum testing. The exploratory test is frequently used because it has a system that fits well with agile principles.

Exploratory testing is software testing in which test cases are not pre-created, but testers check the system instantly. As it can be understood, in the exploratory test, like agile, there is a situation of "unplanned." The focus of exploratory testing is testing as a "thinking" activity.

According to ISTQB, when using “Experience-based Testing” methodologies, test cases are generated based on the tester's previous experience with similar applications and technologies [3], [4]. These strategies can help identify tests that other, more systematic procedures have not been able to identify.

8.8.1 Error guessing

“Error Guessing” is a strategy used in software testing that relies heavily on experience and talent. For example, suppose possible faults and defects are guessed in locations where formal testing would be ineffective. It is also known as experience-based testing, as there is no defined testing technique. Although this is not a formal testing method, it is crucial since it occasionally resolves many outstanding issues.

8.8.2 Exploratory testing

“Exploratory Testing” is a software testing method that is generally defined as continuous learning, test design, and execution. It emphasizes discovery and depends on the individual tester's direction to reveal faults that are not readily addressed in the scope of other tests.

In the past few years, the practice of exploratory testing has grown in popularity. As a result, testers and QA administrators are urged to implement exploratory testing as a thorough test coverage plan.

8.8.3 Checklist-based testing

“Checklist-based Testing” is software testing based on a checklist, which is a planned "to-do" task list. Professional testers with significant technical competence generally prepare these lists. QA professionals use such checklists to guide testing tasks.

Various software systems are frequently examined in the same way to check if it is feasible to discover particular faults or manage functioning circumstances and data without resorting to some systematic approach. Ad-hoc (random) testing refers to such sorts of rudimentary and informal testing. Checklists are used to structure this testing so that time is not wasted repeating the same tasks. Lists of this type might be online, physical, or conceptual.

8.9 Unit Testing

The smallest unit of software design, the software component or module, is the subject of unit testing. The limiting scope of unit testing limits the relative complexity of tests and mistakes discovered. The unit test focuses on internal processing logic and data structures within the bounds of a component. Figure 8.3 modified from [12], depicts a schematic representation of unit testing. Furthermore, the module interface is checked to ensure that data flows appropriately into and out of the software unit under test.

Furthermore, local data structures should be exercised during unit testing, and the local influence on global data should be determined [12]. It is critical to do selective testing of execution pathways during the unit test. Test cases should be written to detect mistakes caused by inaccurate computations, comparisons, or control flow. One of the essential unit testing activities is boundary testing. Unfortunately, the software frequently fails at its limits. For example, when the repeat of a loop with passes is initiated when the maximum or lowest permissible value is met, problems frequently occur when the nth element of an n-dimensional array is handled. Errors are highly likely to be discovered in test scenarios that exercise data structure, control flow, and data values slightly below, at, and just above peaks and minima. When a mistake occurs, an intelligent design anticipates it and offers error-handling mechanisms to reroute or cleanly end processing [12]. Unit testing is typically seen as an insert into the coding process. Unit tests can be designed before or after the source code has been created. A study of design information directs the creation of test cases that are likely to identify problems in each of the previously described categories. Because a component is not a stand-alone application, driver and stub software are frequently required for each unit test. Figure 8.4 modified from [12], depicts the unit test environment. A driver in most applications is just the primary program that receives test-case input, delivers it to the component, and outputs relevant results [12].

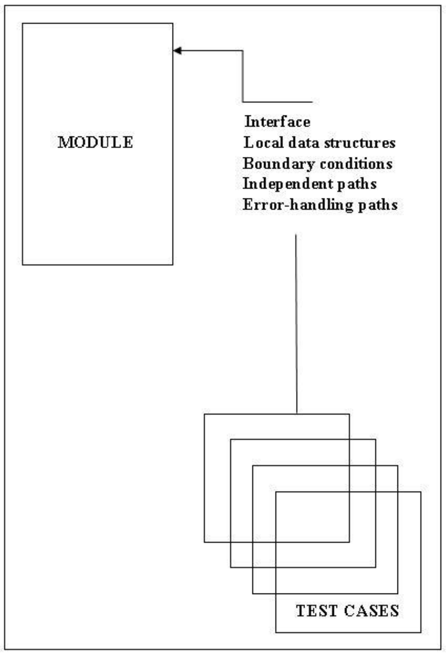


Figure 8.3: Unit test

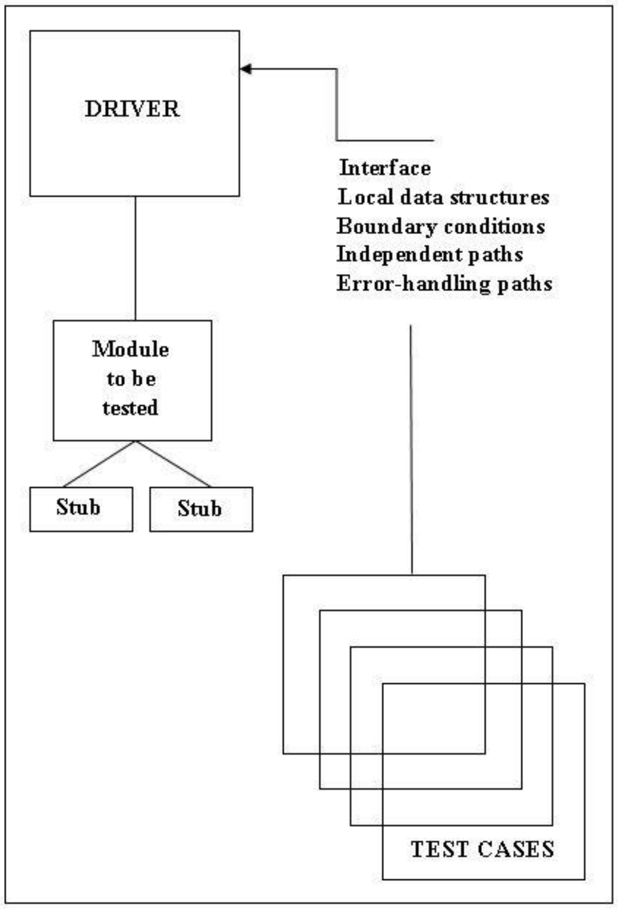


Figure 8.4: Unit test environment

8.10 Integration Testing

All components are combined in advance, and the entire program is tested as a whole. Errors are encountered, but correction is difficult because the isolation of causes is complicated by the vast expanse of the entire program. Several different incremental integration strategies are discussed [12].

8.10.1 Top-down integration

Top-down integration testing is one of the most well-known incremental approaches to constructing the software architecture. Modules subordinate to the central control module are incorporated into the structure in depth-first or breadth-first.

In top-down integration, modules subordinate to the central control module are incorporated into the structure in either a depth-first or breadth-first manner. Also, top-down integration testing is an incremental approach to constructing the software architecture.

8.10.2 Bottom-up integration

Bottom-up integration testing starts with the creation and testing of atomic modules, as the name implies. Because components are integrated from the bottom up, the functionality provided by components that are subordinate to a certain level is always available, eliminating the need for stubs.

8.10.3 Integration testing levels

The following are two stages of integration testing that may be performed on test items of varied sizes [3], [4];

a) Component integration testing is generally performed after component testing and is automated. Component integration tests are usually part of the continuous integration process in iterative and incremental development. External interfaces are not managed by the development organization at the system integration testing level, resulting in various testing difficulties.

b) System integration testing can be performed after or in conjunction with another system testing.

8.10.4 Integration test work products

A test specification contains an overall plan for integrating the program as well as a description of specific tests.

This work product includes a test plan and method, and it is included in the software configuration. Testing is divided into phases, with an emphasis on the product's functional and behavioral features. As a consequence, software builds for each phase are created. A schedule for integration, the generation of overhead software, and other pertinent considerations are also included in the test plan.

8.11 System Testing

During system tests, potential interface problems should be anticipated and error handling paths should be designed that test all information from other elements of the system. A series of tests that simulate insufficient data or other potential errors in the software interface should be performed and recorded.

8.12 Quality-Related Types of Testing

Test techniques are classified as functional and non-functional, as stated before. They aim to produce a quality product. Besides the basic test techniques, other test techniques used in product development are also included in this section. They are classified as follows:

8.12.1 Accessibility testing

The purpose of accessibility testing is to determine whether or not the test item can be handled by persons with a wide range of traits and skills, including those with special accessibility needs. Accessibility testing employs a model of the test item that outlines its accessibility needs and any accessibility design principles to which the test item must adhere.

Accessibility criteria are concerned with a user's capacity to achieve accessibility goals if they have unique accessibility needs. For example, this might require the test item to enable visually and to hear challenged users [8].

8.12.2 Compatibility testing

The purpose of compatibility testing is to determine if the test item can coexist in a shared environment with other independent or dependent (though not necessarily communicative) items. Compatibility testing can also be used on several copies of the same test item or on different items that share a familiar environment [8].

8.12.3 Backup/recovery testing

Recovery testing is a type of system test that involves causing the program to fail in a variety of ways and ensuring that it recovers properly. Re-initialization, check-pointing methods, data recovery, and restart are all examined for accuracy if recovery is automated. If human involvement is required, the mean-time-to-repair is assessed to establish its acceptable bounds [12].

8.12.4 Disaster recovery testing

The goal of disaster recovery testing is to establish if the test item's operation can be transferred to another operational location in the case of a failure and whether it can be transferred back once the fault has been resolved. Disaster recovery testing employs a test item model that outlines its disaster recovery criteria, including any needed disaster recovery design guidelines to which the test item must adhere. In disaster recovery testing, the test item might be a whole functioning system, complete with buildings, staff, and processes.

For example, disaster recovery testing may include processes performed by operational people, data migration, software, employees, offices, or other facilities, or retrieving data already backed up to a remote site [8].

8.12.5 Deployment testing

Deployment testing, also known as configuration testing, puts the program through its tests in each environment in which it will be used.

Furthermore, deployment testing looked at all installation methods. It is specialized installation software that customers will use and any documentation used to present the product to end-users [12].

8.12.6 Installability testing

The goal of installability testing is to establish whether a test item can be installed, uninstalled/removed, and upgraded as needed in all defined situations.

Installability testing employs a model of the test item's installability necessities, which are typically indicated in the installation, uninstallation, or upgrade processes, who will carry out the installation, uninstallation, or upgrade, the target systems. The test items must be installed, uninstalled, or upgraded [8].

8.12.7 Interoperability testing

The goal of interoperability testing is to evaluate if a test item can interact successfully with many other test items or systems in the same or other contexts, including whether the test item can effectively utilize information received from other systems. Interoperability testing employs a model of the test item that outlines its interoperability criteria, including interoperability design standards to which the test item must adhere; this might involve determining if a test item operating in one environment can communicate correctly with another or a system operating in a different environment [8].

8.12.8 Localization testing

The goal of localization testing is to verify whether the test item can be comprehended within the geographical location in which it is required to be utilized. Localization testing can involve (but is not limited to) determining if the user interface and accompanying documentation of the test item can be comprehended by users in each nation or region of usage [8].

8.12.9 Maintainability testing

The purpose of maintainability testing is to determine whether or not a test item can be maintained up with a fair amount of labor. Maintainability testing uses a model of the test item's maintainability requirements, which are frequently stated in terms of the effort required to make a change in the categories of corrective, perfective, adaptive, and preventive maintenance [8].

8.12.10 Portability testing

Portability testing aims to assess the ease or difficulty with which a test item may be transported from one hardware, software, or other operational or usage environment to another.

Portability testing employs a model of the test item that outlines its portability requirements and any mandatory portability design criteria to which the test item must adhere. For example, the capacity to transfer the test item from one environment to another, or to change the configuration of the current environment to other needed settings, is addressed by portability criteria [8].

8.12.11 Performance testing

Performance testing aims that while tests are being run, the performance of a single module may be reviewed. However, the actual performance of a system cannot be determined until all system parts are fully integrated. Therefore, performance tests are frequently combined with stress tests and need hardware and software instrumentation. The tester can identify conditions contributing to deterioration and possible system failure by instrumenting a system [12].

8.12.12 Stress testing

The goal of stress testing is for the tester to try to break the software. In some cases, a tiny range of data inside a program's legitimate data boundaries might result in excessive and even erroneous processing, as well as significant performance deterioration. Stress testing aims to find data combinations within valid input classes that might lead to instability or incorrect processing [12].

8.12.13 Security testing

Any computer-based system that maintains sensitive information or performs activities that may inadvertently hurt humans is a potential target for inappropriate or unlawful intrusion. Security testing ensures that the protective measures placed into a system will protect it from unauthorized access. The system designer's responsibility is to make penetration more expensive than the value of the information gained [12].

8.12.14 Usability testing

Usability testing aims to determine if specific users can use the test item to fulfill given goals effectively, efficiently, and satisfactorily in specified usage contexts.

Usability testing, as a result, employs a model of the test item that outlines its usability criteria and any needed usability design standards to which the test item must adhere.

Usability criteria define the test item's usability goals. Usability objectives must be based on test item objectives and contexts of usage for the test item. Usability goals will be established to improve the effectiveness, efficiency, and satisfaction of specified users in achieving stated goals in one or even more specified contexts of the use [8].

8.12.15 Smoke testing

Smoke testing is a sort of integration testing that is frequently used while building software for products. Its purpose is to act as a pacing mechanism for time-sensitive projects, allowing the software team to assess the project on a frequent basis. The goal should be to find show-stopper problems that are most likely to cause the software project to fall behind schedule. The daily testing frequency allows managers and practitioners to measure integration testing progress in a realistic manner. Because smoke tests are conducted on a frequent basis, incompatibilities and other show-stopping faults are detected early, reducing the risk of substantial schedule disruption. As a result, the quality of the finished product has improved. Smoke testing is likely to identify functional, architectural, and component-level design issues since it is a construction-oriented technique. If these flaws are fixed as quickly as feasible, product quality will improve. As a result, additional software is being incorporated and tested. [12].

8.13 Acceptance Testing

This testing, like system testing, is frequently concerned with a system's or product's general behavior and capacity. According to the ISTQB, the goals of this testing to create confidence in the system's overall quality. Confirming that the system's functional and non-functional behaviors are as described [3], [4].

“Acceptance Testing” may provide information that may be utilized to determine whether the system is ready for deployment and use by the customer. Although this is not usually the intention, problems can be identified during acceptance testing. Finding many defects during acceptance testing might be considered a severe project risk in some instances.

The following are examples of common types of acceptability testing:

- User acceptance testing;
- Operational acceptance testing;

- Contractual & regulatory acceptance testing;
- Alpha & beta testing.

In the following subsections, they are briefly summarized [3], [4]:

8.13.1 User acceptance testing

In a simulated or virtual operating environment, “User Acceptance Testing” evaluates a system's fitness for usage by intended users. The fundamental goal, according to the ISTQB, is to provide users confidence that they can use the system to satisfy their needs, meet requirements, and conduct business activities with the least amount of complexity, expense, and risk.

8.13.2 Operational acceptance testing

Employees from operations or administration frequently test acceptance systems in a (simulated) production environment. The tests are centered on operational aspects and might include the following:

- Backup and restoration testing;
- Installing, removing, and upgrading;
- Disaster recovery;
- User administration Maintenance tasks;
- Data loading and migration jobs;
- Checks for security flaws;
- Performance testing

The primary goal of operational acceptance testing is to inspire confidence in the operators or system administrators that the system will continue to function.

8.13.3 Contractual & regulatory acceptance testing

“Contractual Acceptance Testing” assures that custom-developed software satisfies the contract's acceptance criteria. Acceptance criteria should be defined when the parties sign the contract. Users or independent testers typically perform contractual acceptability testing.

“Regulatory Acceptance Testing” is conducted following applicable rules, such as regulatory, legal, or safety requirements. Users or independent testers commonly conduct regulatory acceptability testing, with regulatory agencies occasionally seeing or reviewing the results.

“Contractual & Regulatory Acceptance Testing” primary purpose is to ensure that contractual or regulatory compliance has been achieved.

8.13.4 Alpha & Beta testing

“Alpha Testing” is done on the premises of the developing firm by potential or current consumers, operators, or an independent test team, rather than by the development team. Consumers and operators, both potential and current, do Beta testing on their premises. “Beta Testing” might take place after or without any preceding alpha testing. The purpose of Alpha and Beta testing is to provide prospective customers and operators confidence that they will use the system in real-world, day-to-day situations and operational contexts to achieve their goals with the least amount of hassle, money, and danger. Another purpose may be to find flaws in the situations and surroundings in which the system will be used, especially if the development team finds it impossible to replicate such conditions and environments. Acceptance tests are done when custom software is produced for a specific customer to check that all requirements are satisfied. An acceptance test, which is carried out by the end-user rather than software engineers, can range from a casual test drive to a thoroughly planned and executed series of tests. If the software is built as a product that will be used by a large number of people, formal acceptance testing is impossible. Instead, most software product developers utilize alpha and beta testing to discover vulnerabilities that only the end-user appears to be capable of identifying. The program is put to the test in a real-world situation. The Beta test is conducted at one or more end-user sites. Client acceptance testing, a sort of beta testing, is occasionally performed when customized software is given under a contract. Before adopting the developer's work, the customer conducts a series of tests to find defects [12].

9. AN APPLICATION

9.1 General

The “Test Plan” is intended to specify the scope, strategy, resources, and timeline for all testing activities associated with the project according to “Guru99 Banking Project” [17]. The test plan specifies the things to be tested, the features to be tested, the types of testing to be conducted, the persons in charge of testing, the resources and timeline needed to complete testing, and the risks connected with the plan. Table 9.1 modified from [17], shows the project log data.

Table 9.1: Change Log

VERSION #	**REVISION DATE**	**WRITTEN BY**	**DESCRIPTION**	**APPROVAL DATE**
1.0	12/14/2021	Bugra	Test plan created.	12/18/2021
1.1	12/15/2021	Bugra	Test plan updated.	12/19/2021

9.2 Scope of Testing

This “Project's Scope” is limited to testing the features indicated in the following parts of this document. Non-functional testing, such as stress and performance, is outside this project's scope. Automation testing is outside the scope of this document. Functional testing and external interfaces are included in the scope and must be tested. The banking site will only be compatible with Google Chrome Versions 27 and higher (compatibility testing) for non-functional requirements. Table 9.2 modified from [18], shows the project modules representing functional requirements. The following functional requirements include the features that this project, which is considered an example, must fulfill and are considered requirements by the customer. As mentioned in the thesis, at this stage, the analysis of the features that can and cannot be tested using static test methods will be done at this stage, as well as the test design will be started at the same time. Also, Table 9.3 modified from [18], contains the SRS plan for the project in detail. This table lists the requirements output features of the sample project in detail.

Table 9.2: Description of the modules (functional requirements)

MANAGER	**CUSTOMER**
“New Customer”	“Balance Enquiry”
“Edit Customer”	“Fund Transfer”
“Delete Customer”	“Mini Statement”
“New Account”	“Customized Statement”
“Edit Account”	“Change Password”
“Delete Account”	“Login & Logout”
“Deposit”	
“Withdrawal”	
“Fund Transfer”	
“Change Password”	
“Balance Enquiry”	
“Mini Statement”	
“Customized Statement”	
“Login & Logout”	

Table 9.3: Software Requirement Specifications

MODULE NAME	APPLICABLE ROLES	DESCRIPTION
“Fund Transfer”	“Manager” “Customer”	“Customer: A customer can request that monies be transferred from the one’s "own" account to any other account. Manager: A manager can move funds from any source bank account to any destination bank account”.
“Mini Statement”	“Manager” “Customer”	“A Mini Statement displays the latest five transactions of an account. Customer: A consumer can only view Mini Statement for the one’s "own" accounts. Manager: Any account's Mini Statement can be viewed by a manager”.
“Customized Statement”	“Manager” “Customer”	“A customized statement allows one to filter and show transactions in an account depending on the date and transaction value. Customer: A customer can only access Customized Statements for the one’s "own" accounts. Manager: Any account's Customized Statement can be seen by a manager”.
“Change Password”	“Manager” “Customer”	“Customer: Only the customer's account password can be changed. Manager: A manager can only change the password for the one’s account. One unable to alter the one’s clients' passwords”.

“New Customer”	“Manager”	“Manager: A manager has the authority to add a new customer. Manager: A manager can change a customer's address, email, and phone number”.
“New Account”	“Manager”	“Currently, the system offers two types of accounts: • Saving • Current A customer may have many savings accounts. The one can have many current accounts for different businesses. The one can also have numerous current and savings accounts. Manager: A manager can create a new account for an existing client”.
“Edit Account”	“Manager”	“Manager: A manager can add or update account data for an existing account”.
“Delete Account”	“Manager”	“Manager: A manager can create or remove a customer's account”.
“Delete Customer”	“Manager”	“A customer can only be removed if one’s has no active current or savings accounts. Manager: A manager has the authority to remove a customer”.
“Deposit”	“Manager”	“Manager: A manager has the authority to put funds into any account. When cash is deposited in a bank branch, this is usually done”.
“Withdrawal”	“Manager”	“Manager: A manager has the authority to remove money from any account. When cash is withdrawn from a bank branch, this is usually done”.

9.3 Features Not To Be Tested

These features will not be examined since they are not listed in the software requirements:

- User interfaces,
- Hardware interfaces,
- Software interfaces,
- Database logical communications interfaces,
- Website security and performance.

9.4 Test Type

Three forms of testing will be carried out. First, individual software components are merged and evaluated as a group during integration testing. System testing is performed on a fully integrated system to assess the system's conformance with the defined requirements. Finally, “API Testing” validates all APl's written for the product under test.

The results of the three main types of tests to be performed will pass through the sequential stages as shown in the Test Activities & Tasks section and enter the Test Completion section, which is the last activity and will be terminated after the test.

9.5 Risk and lssues

Table 9.4 modified from [17], shows the risks and issues of the project. Risks and mitigations constitute an important content of the errors and root cause analysis section, which are explained in detail in the text of the thesis. When errors are detected, and root causes are revealed, it will be easier and more effective to produce a quality software output.

9.6 Quality Objective

Quality Objective is the primary objective that must be met without testing. Ensure that the “AUT (Application Under Test)” adheres to functional and non-functional criteria. Ascertain that the AUT fulfills the client's quality standards.

Table 9.4 shows risks and issues about the project. Before releasing the system to the public, bugs and issues are detected and corrected.

Table 9.4: Risk and Issues

RISK	**MITIGATION**
“Team members lack the knowledge and skills required for specific testing. Therefore, the tests may not be run on time by the personnel in the institution, and in this case, a third-party software testing service may be required”.	“Plan a training course to help one team members improve their skills”.
“The project timetable is overly constrained; it will be challenging to accomplish this job on time”.	“Set the Test Priority for each test activity”.
“Test Manager shows a lack of management abilities”.	“Plan leadership development for the manager”.
“Employee productivity declines as a result of ineffective collaboration”.	“Encourage each team member in their assignment, and workers' productivity motivates them to work more”.
“Overestimation of the budget and expenditure overruns”.	“Establish the project's scope before commencing work, pay close attention to project planning, and track and measure progress regularly”.

9.7 Roles and Responsibilities

A detailed explanation of each team member's tasks and responsibilities, such as;

- Test Analyst,
- Test Manager,
- Developers,
- Project Manager.

9.8 Test Methodology

The project's testing technique might be;

- Waterfall,
- Iterative,
- Agile (Extreme Programming (XP), Scrum, LAPIS).

The methodology to be selected depends on multiple factors.

9.9 Test Levels

The type of testing done on the AUT is defined by the Test Levels. The Testing Levels are essentially determined by the scope, time, and money restrictions of the project.

The purpose of the tests is to check and verify the project's functionality. As a result, the project should concentrate on testing banking functions such "Account Management", "Withdrawal", and "Balance".

Table 9.3 ensures that all of these activities go well in a real-world corporate environment.

9.10 Bug Triage

The bug triage's purpose is to specify the kind of resolution for each problem, prioritize bugs, and set a plan for all "To Be Fixed Bugs".

Because certain faults may be critical for the system and usage, errors should be addressed in the order that they occur, taking into account the frequency of error occurrence and the severity of the issue.

Bug Triage is crucial in the example application.

9.11 Resource & Environment Needs

Make a list of tools like:

- Requirements tracking tool,
- Bug tracking tool,

Table 9.5 modified from [17], shows resource & environmental needs,

Table 9.5: Resource & Environment Needs

RESOURCES	DESCRIPTIONS
“Servers Specifications”	“MySQL server is installed by the database server, whereas Apache server is installed by the web server”.
“Test Tool Specifications”	“Create a test tool that will automatically create test results in a predefined format and run automated tests”.
“Networks Specifications”	“Set up a gigabit LAN and one internet line with a minimum speed of 5 Mb/s”.
“Computers Specifications”	“At least four PCs, each with Windows 8, 2GB of RAM, and a 3.4GHz processor”.

9.12 Suspension Criteria and Resumption Requirements

Suspension criteria provide the conditions under which all or part of the testing procedure can be halted, whereas resumption criteria specify when testing can continue once it has been interrupted.

9.13 Test Completeness

This is where the criteria for concluding the testing are defined.

For example, one hundred percent test coverage, all human and automated test cases performed, and all open problems repaired or fixed in the future release are a few criteria to assess test completeness.

9.14 Test Deliverables

Here one should include all of the test artifacts that were provided during the different stages of the testing lifecycle. The following are some examples of deliverables;

9.14.1 Before testing phase

- Test plan,
- Test cases,
- Test design specifications.

9.14.2 During the testing phase

- Test tools,
- Test simulators,
- Test data's,
- Test traceability matrix,
- Error logs and execution logs.

9.14.3 After the testing cycle is over

- Test results and reports,
- Defect report,

- Installation and test procedures guidelines,
- Release notes.

9.15 Test Environment

It just includes the bare minimum of hardware for testing the program.

In addition to client-specific apps, the following software tools are required.

- Windows 8 and later versions,
- Office 2013 and later versions,
- MS Project 2013 and later versions.

10.CONCLUSIONS

In this thesis, available resources about software testing was compiled firstly, and the relationship between software testing and software quality was studied. Next, “ISO/IEC/IEEE 29119 Standards” which are in six parts, were collected. Part 6 of this series of standards was published quite recently in 2021. It includes guidelines for the use of “ISO/IEC/IEEE 29119 Standards” in agile projects.

ISTQB (International Software Testing Qualification Board) is an international organization closely related to software testing. A brief section is devoted to ISTQB in the thesis.

Software Testing Principles, Test Processes, and Test Techniques are discussed in detail and summarized after going through the relevant parts of “ISO/IEC/IEEE 29119 Standard”.

The major contribution of the thesis may be stated as an Application Project, to be developed referring to the parts of “ISO/IEC/IEEE 29119” Standard. For the Application Project the Software Requirements Specification (SRS) document of a sample open-source project is used successfully.

It is believed that the results of the thesis study may become more useful in the future with different applications as a continuation of the thesis study.

One of the most difficult issues in the study is the limited resources in this field and the adherence to the own publications and paid courses of a few authors who have made a fame in the international market.

REFERENCES

[1] *Software and systems engineering - Software testing - Part 1: Concepts and definitions*, TSE/ISO/IEC/IEEE 29119-1, Switzerland, Sept. 2013

[2] R. Black, *Advanced Software Testing.* / Vol. 1, Guide to the ISTQB Advanced Certification as an Advanced Test Analyst, 1st ed., vol. 1. Santa Barbara, CA: Rocky Nook, 2009, pp. 104–106.

[3] "Certified Tester Foundation Level Syllabus," International Software Testing Qualifications Board, Nov. 2019. Accessed: Feb. 20, 2021. [Online]. Available: https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html.

[4] "Sertifikalı Test Uzmanı Temel Seviye Ders Programı Versiyon 2018 International Software Testing Qualifications Board," Yazılım Test ve Kalite Derneği, Turkish Testing Board, Dec. 2019. Accessed: Feb. 20, 2021. [Online]. Available: https://www.turkishtestingboard.org/foundation-level-ders-programi-turkce/.

[5] "ASTQB – American Software Testing Qualifications Board - Software Testing Certification - ISTQB Exam Registration," *ASTQB*, 2019. https://astqb.org/ (Accessed Nov. 07, 2019).

[6] *Software and systems engineering - Software testing - Part 2: Test processes,* TSE/ISO/IEC/IEEE 29119-2, Switzerland, Sept. 2013

[7] *Software and systems engineering - Software testing - Part 3: Test documentation,* TSE/ISO/IEC/IEEE 29119-3, Switzerland, Sept. 2013

[8] *Software and systems engineering - Software testing - Part 4: Test techniques,* TSE/ISO/IEC/IEEE 29119-4, Switzerland, Dec. 2015

[9] *Software and systems engineering - Software testing - Part 5: Keyword-Driven Testing*, TSE/ISO/IEC/IEEE 29119-5, Switzerland, Nov. 2016

[10] *Software and systems engineering - Software testing - Part 6: Guidelines for the use of ISO/IEC/IEEE 29119 (all parts) in agile projects*, TSE/ISO/IEC TR 29119-6, Switzerland, July 2021

[11] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, Eight. New York, NY: McGraw-Hill Education, 2015, pp. 413–428.

[12] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, Eight. New York, NY: McGraw-Hill Education, 2015, pp. 466–493.

[13] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, Eight. New York, NY: McGraw-Hill Education, 2015, pp. 541–563.

[14] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, Eight. New York, NY: McGraw-Hill Education, 2015, pp. 497–521.

[15] C. Lal et all., "Boundary Value Analysis and equivalence class partitioning with simple example," *Software Testing Class*, 11-Nov-2013. [Online]. Available: https://www.softwaretestingclass.com/boundary-value-analysis-and-equivalence-class-partitioning-with-simple-example/. [Accessed: 21-Dec-2021].

[16] A. Gürbüz, *Yazılım Test Mühendisinin El Kitabı*. Ankara: Seçkin Yayıncılık, 2020.

[17] "Test plan GURU99," *studylib.net*, 15-Feb-2019. [Online]. Available: https://studylib.net/doc/25218654/test-plan-guru99. [Accessed: 16-Dec-2021].

[18] T. Hamilton, "Banking Domain Application Testing: Sample test cases," *Guru99*, 08-Oct-2021. [Online]. Available: https://www.guru99.com/banking-application-testing.html. [Accessed: 16-Dec-2021].