CHAPTER – 12
SYSTEM TEST PLANNING AND EXECUTION

OUTLINE OF THE CHAPTER

STRUCTURE OF A SYSTEM TEST PLAN

The purpose of a system test plan is as follows:

It provides guidance for the executive management to support the test project

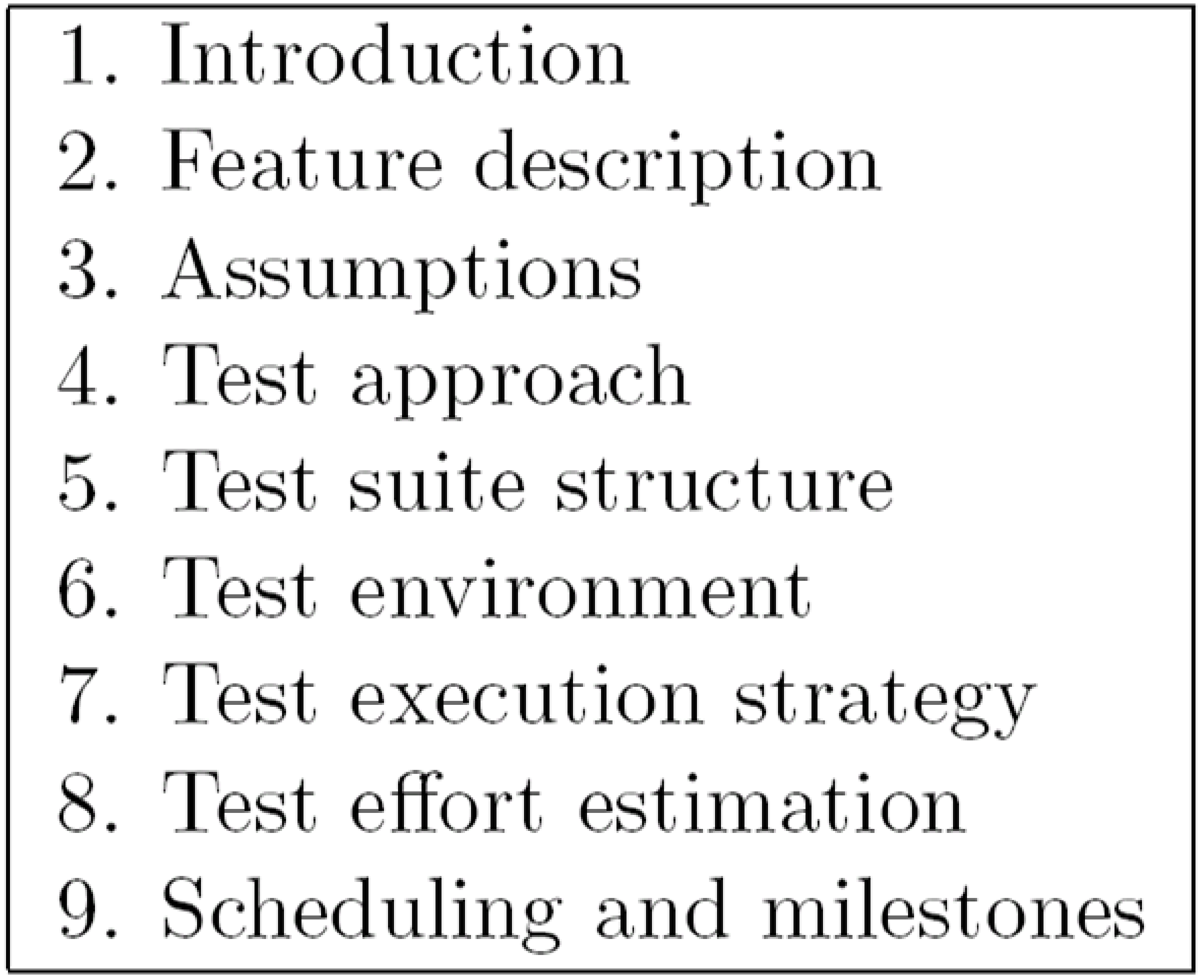
It establishes the foundation of the system testing part of the overall software project

It provides assurance of test coverage by creating a requirement traceability matrix

It outlines an orderly schedule of events and test milestones that are tracked

It specifies the personnel, financial, equipment, and facility resources required to support the system testing part of a software project

STRUCTURE OF A SYSTEM TEST PLAN



1. Introduction
2. Feature description
3. Assumptions
4. Test approach
5. Test suite structure
6. Test environment
7. Test execution strategy
8. Test effort estimation
9. Scheduling and milestones

12.1: An outline of a system test plan.

INTRODUCTION AND FEATURE DESCRIPTION

The *introduction* section of the system test plan includes:

- Test project name
- Revision history
- Terminology and definitions
- Name of the approvers and the date of approval
- References
- Summary of the rest of the test plan

The *feature description* section summarizes the high level description of the functionalities of the system

TEST APPROACH

The *test approach* section describes the following aspect of the testing project

- Issues discovered by customers that were not caught during system testing in the past project are discussed and the preventive action that are being taken in this test project
- If there are any outstanding issues that need to be tested differently need to be discussed here
- A test automation strategy for writing scripts is a topic of discussion
- Identify test cases from the database that can be re-used in this test plan
- Give an outline of the tools, formats, and organizing scheme, such as traceability matrix that will be used and followed during the test project

TEST SUITE STRUCTURE

Detail *test groups* and *subgroups* are outlined based on the test categories identified in the *test approach* section

Test objectives are created for each test group and subgroups based on the system requirements and functional specification documents

Identification of test objectives provides a clue to the total number of test cases needs to be developed

A traceability matrix is generated to make an association between requirements and test objectives to provide the test coverage

TEST ENVIRONMENT

Multiple test environments are constructed in practice

- To run scalability tests one need more resources than to run functionality tests
- To reduce the length of testing time

A schematic diagram of one or more test beds are presented in this section of the system test plan

- A high-level graphic layout of the test architectures
- A table of types of equipments, their quantities, and their descriptions to support the test architecture

TEST EXECUTION STRATEGY

The processes of system test execution, defect detection, and fixing defects are intricately intertwined

The key characteristics of those processes are as follows.

- Some test cases cannot be executed unless certain defects are detected and fixed
- A programmer may introduce new defects while fixing one defect, which may not be successful
- The development team releases a new build for system testing by working on a subset of the reported defects, rather than all the defects
- It is a waste of resources to run the entire test set T on a build if too many test cases fail

A MULTI-CYCLE SYSTEM TEST STRATEGY

- The objective of the first test cycle is to detect most of the defects by executing all the test cases.
- The objective of the second test cycle is to verify the fixes for the defects found in the first test cycle.
- The objective of the third and final test cycle is to ensure that the software is stable before it can be released to the customer.

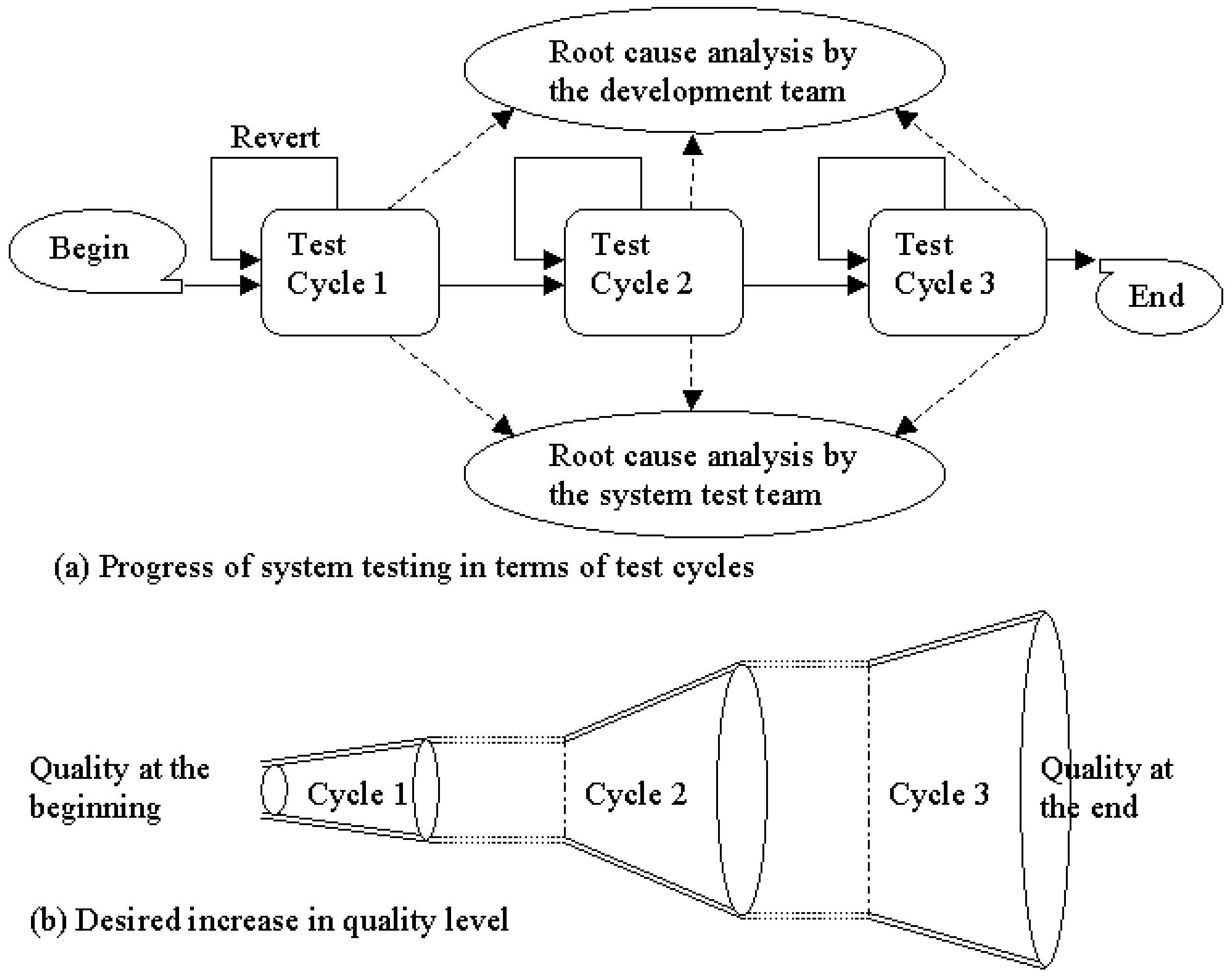

Root cause analysis by the development team
Revert
Begin
Test Cycle 1
Test Cycle 2
Test Cycle 3
End
Root cause analysis by the system test team
(a) Progress of system testing in terms of test cycles
Quality at the beginning
Cycle 1
Cycle 2
Cycle 3
Quality at the end
(b) Desired increase in quality level

Figure 12.1: The concept of a cycle-based test execution strategy

CHARACTERIZATION OF TEST CYCLES

Each test cycle is characterized by a set of six parameters:

- Goals
- Assumptions
- Test execution
- Revert and Extension criteria
- Actions
- Exit criteria

CHARACTERIZATION OF TEST CYCLES

Goals
- System test team sets its own goals to be achieved in each test cycle
- These goals are ideal in the sense these are very high standard
- Goals are specified in terms of the number of test cases to pass in a cycle

Assumptions
- How often the builds will be selected during in a system test cycle?
- For example “the team can accept builds from SIT group on a daily basis during a test cycle”

Test Execution
- Prioritization of test execution changes between test cycles
- Test cases that exercise basic functionalities have higher priority than the rest in the first test cycle
- Test cases that have failed in one test cycle have a higher priority in the following test cycle
- Test cases in certain groups have higher priority than others

CHARACTERIZATION OF TEST CYCLES

Revert and Extension criteria

- It may not be useful to continue a test cycle if it is found that a software is of poor quality
- Often a test cycle is extended due to various reasons
  - A need to re-execute all the test cases in a particular test group because a large fraction of the test cases within the group failed
  - A significantly large number of new test cases were added while test execution was in progress
- The conditions for prematurely terminating a test cycle and for extending a test cycle must be precisely stated

CHARACTERIZATION OF TEST CYCLES

Actions

- Too many test cases may fail during a test cycle
  - The development team is alerted
  - The developers take action in the form of root cause analysis (RCA)
  - Corrective actions are taken by updating the design specification, reviewing the code, and adding new test cases to the unit and integration test plan
- The system test team has to design a large number of new test cases during a test cycle
  - The system test team initiates a root cause analysis (RCA)
  - The team studies the new test cases and categories them into different groups based on the functional requirements
  - The relevant requirements are studied to understand why the test team was unable to identify the objectives of these new test cases in the first place

CHARACTERIZATION OF TEST CYCLES

Exit Criteria

- An exit criterion specifies the termination of a test cycle
- One may exit a test cycle even if goals for that particular test cycle are not fully achieved
- The exit criteria are defined based on few quality metrics associated with the test cycle
- These quality metrics must be monitored during the test cycle
- Example of exit criteria
  - 95% of test cases passed and all the known defects are in CLOSED state

PRIORITIZATION OF TEST CASES

Prioritization of test cases means ordering the execution of test cases according to certain test objectives

Formulating test objectives for prioritization of individual test cases is an extremely difficult task

In system testing, test cases are prioritized in terms of groups with common properties

In a multi-cycle based test execution strategy, test cases are prioritized in a group in each test cycles for three reasons:

- initially the quality level of the system under test is not very high
- the quality of the system keeps improving from test cycle to test cycle
- a variety of defects are detected as testing progresses

TEST EFFORT ESTIMATION

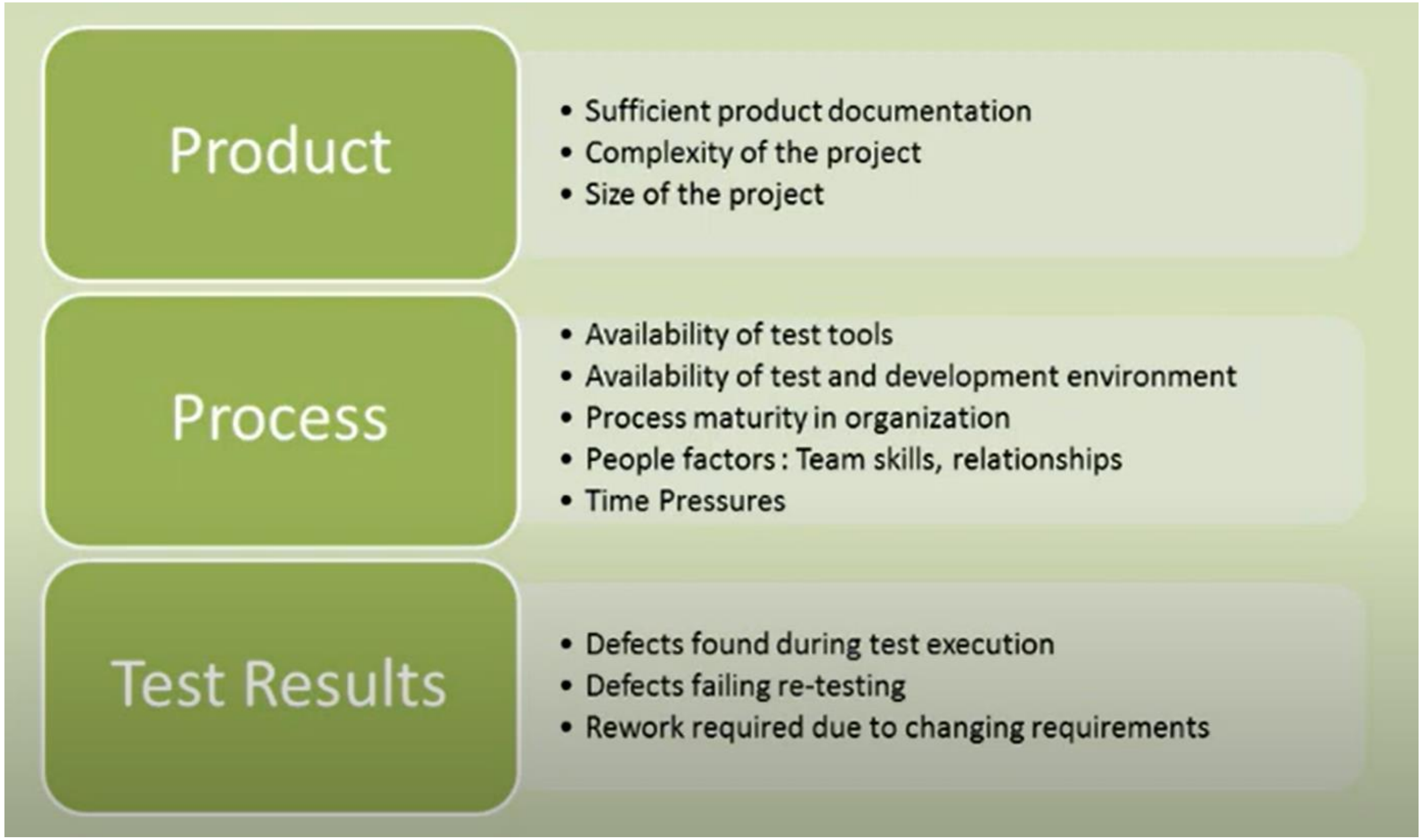
The system test group needs to estimate testing effort to produce a schedule of test execution. Intuitively, testing effort defines the amount of work that needs to be done.

Two major components:

The number of test cases created by one person in one day

The number of test case executed by one person in one day

FACTORS AFFECTING TEST EFFORT



Product

- Sufficient product documentation
- Complexity of the project
- Size of the project

Process

- Availability of test tools
- Availability of test and development environment
- Process maturity in organization
- People factors : Team skills, relationships
- Time Pressures

Test Results

- Defects found during test execution
- Defects failing re-testing
- Rework required due to changing requirements

TEST EFFORT ESTIMATION

Estimation of Number of test cases

- Estimation of Number of Test Cases Based on Test Group Category
  - It is straightforward to estimate the number of test cases after the test suite structure and the test objectives are created
  - Simply count the number of test objectives
- Estimation of Number of Test Cases Based on Function Points
  - Total number of test cases = $(\text{Function Points})^{1.2}$

FUNCTION POINTS

The central idea in the function point method is as follows:

Given a functional view of a system, in the form of the number of user inputs, the number of user outputs, the number of user on-line queries, the number of logical files, and the number of external interfaces, one can estimate the project size in number of lines of code required to implement the system and the number of test cases required to test the system

The function point of a system is a weighted sum of the numbers of

- inputs,
- outputs,
- master files
- inquiries produced to, or generated by, the software

COMPUTATION OF FUNCTION POINT

Step 1: Identify the following five types of components, also known as “user function types,” in a software system

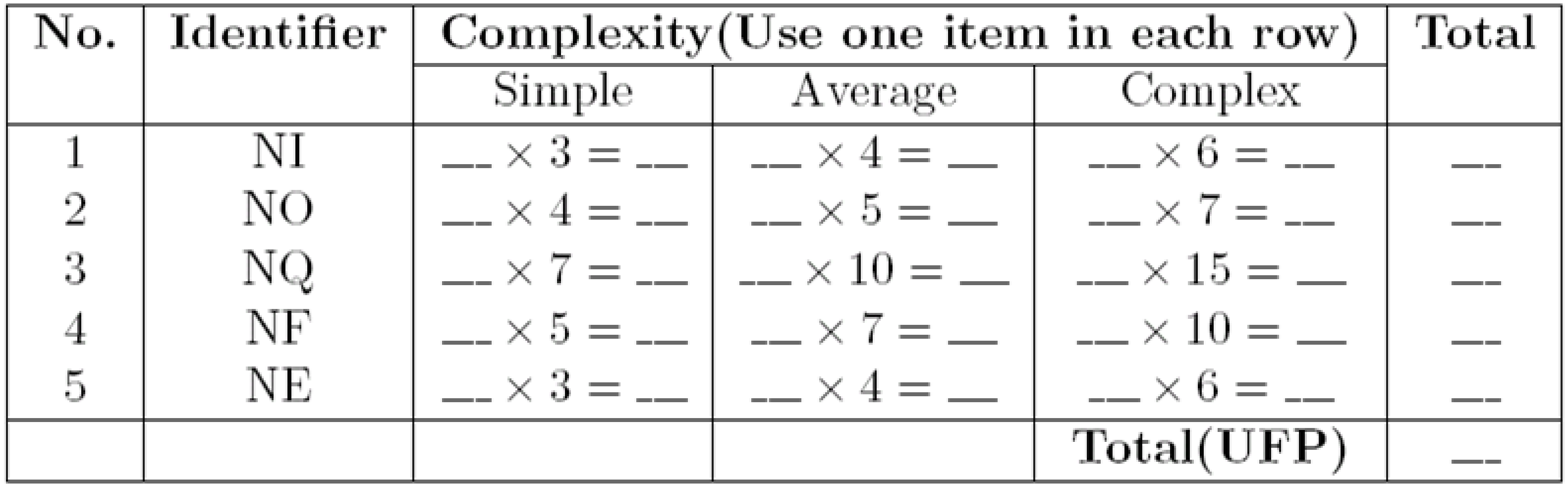
- Number of external input types (NI)
- Number of external output types (NO)
- Number of external inquiry types (NQ)
- Number of logical internal file types (NF)
- Number of external interface file types (NE)

COMPUTATION OF FUNCTION POINT

Step 2: Analyze the complexity of each of the above five types of user function and classify those to three levels of complexities, namely *simple*, *average*, or *complex*

Compute the unadjusted function point (UFP) by using the form shown in Table 12.7

UFP = WFNI × NI + WFNO × NO + WFNQ × NQ + WFNL × NL + WFNE × NE



No.	Identifier	Complexity(Use one item in each row)			Total
		Simple	Average	Complex	
1	NI	__ × 3 = __	__ × 4 = __	__ × 6 = __	__
2	NO	__ × 4 = __	__ × 5 = __	__ × 7 = __	__
3	NQ	__ × 7 = __	__ × 10 = __	__ × 15 = __	__
4	NF	__ × 5 = __	__ × 7 = __	__ × 10 = __	__
5	NE	__ × 3 = __	__ × 4 = __	__ × 6 = __	__
				Total(UFP)	__

Table 12.7: A form for computing the unadjusted function point.

COMPUTATION OF FUNCTION POINT

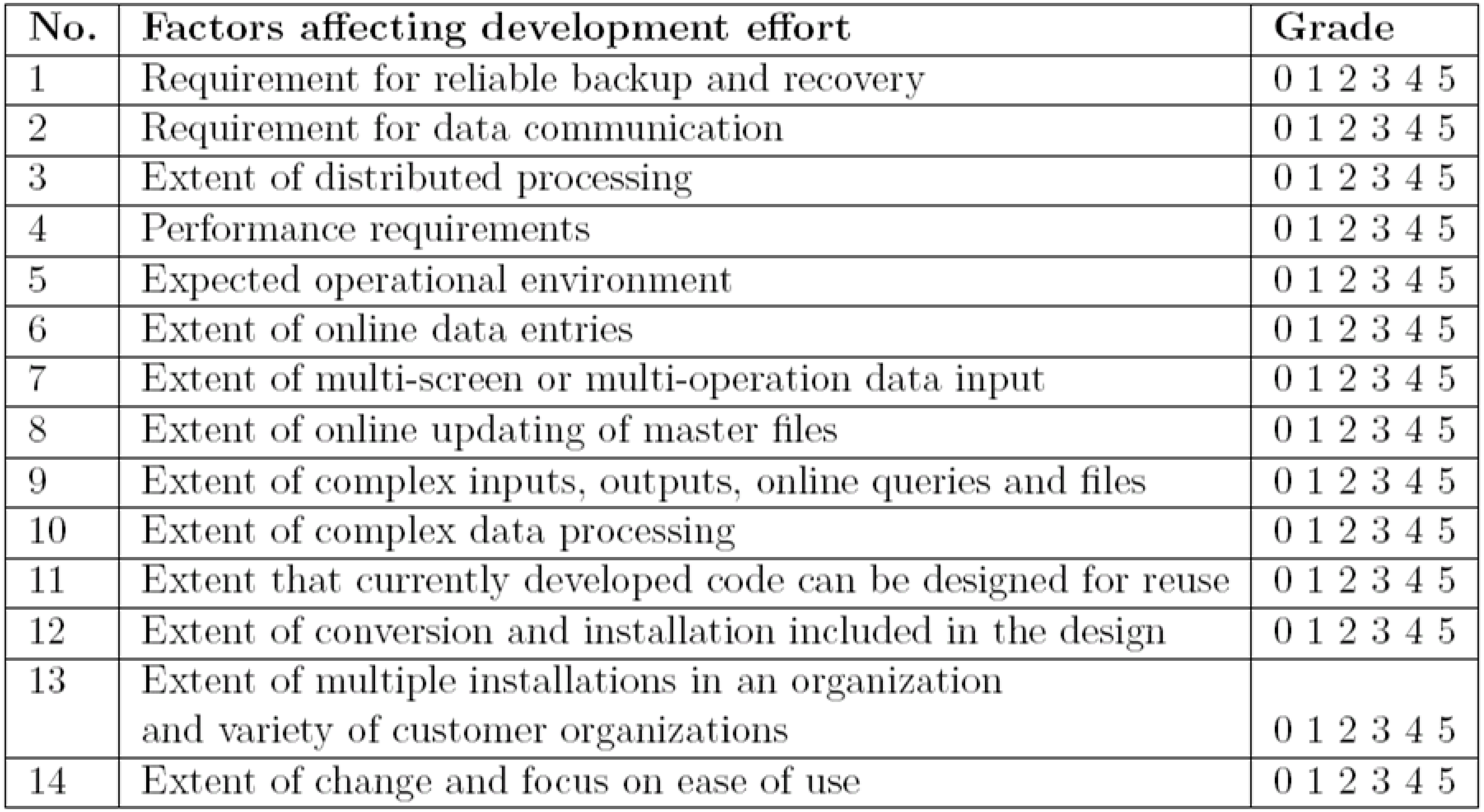
Step 3: Identify 14 factors that affect the required development effort for a project. A grade – between 0 and 5 – is assigned to each of the 14 factors for a certain project.

The sum of these 14 grades is known as processing complexity adjustment (PCA) factor. The 14 factors have been listed in Table 12.8

In Table 12.8, the degree of influence of a factor on development effort is measured as follows:

- Not present, or no influence if present = 0
- Insignificant influence = 1
- Moderate influence = 2
- Average influence = 3
- Significant influence = 4
- Strong influence = 5

COMPUTATION OF FUNCTION POINT



No.	Factors affecting development effort	Grade
1	Requirement for reliable backup and recovery	0 1 2 3 4 5
2	Requirement for data communication	0 1 2 3 4 5
3	Extent of distributed processing	0 1 2 3 4 5
4	Performance requirements	0 1 2 3 4 5
5	Expected operational environment	0 1 2 3 4 5
6	Extent of online data entries	0 1 2 3 4 5
7	Extent of multi-screen or multi-operation data input	0 1 2 3 4 5
8	Extent of online updating of master files	0 1 2 3 4 5
9	Extent of complex inputs, outputs, online queries and files	0 1 2 3 4 5
10	Extent of complex data processing	0 1 2 3 4 5
11	Extent that currently developed code can be designed for reuse	0 1 2 3 4 5
12	Extent of conversion and installation included in the design	0 1 2 3 4 5
13	Extent of multiple installations in an organization and variety of customer organizations	0 1 2 3 4 5
14	Extent of change and focus on ease of use	0 1 2 3 4 5

Table 12.8: Factors affecting development time.

COMPUTATION OF FUNCTION POINT

Step 4: Compute the function points (FP) of a system using the following empirical expression:

$$FP = UFP \times (0.65 + 0.01 \times PCA)$$

- By multiplying the unadjusted function point, denoted by UFP, by the expression (0.65 + 0.01 × PCA), we get an opportunity to adjust the function point by +/- 35 percent.
- Let us analyze the two extreme values of PCA, namely 0 and 70 (= 14 × 5).
  - PCA = 0 : FP = 0.65 × UFP
  - PCA = 70 : FP = UFP × (0.65 + 0.01 × 70) = 1.35 × UFP
- The value of FP can range from 0.65 × UFP to 1.35 × UFP
- The value of FP is adjusted within a range of +/- 35 percent by using intermediate values of PCA.

EXERCISE 1

Consider the following system:

User wants to maintain customer data and product data and needs to reference supplier data.

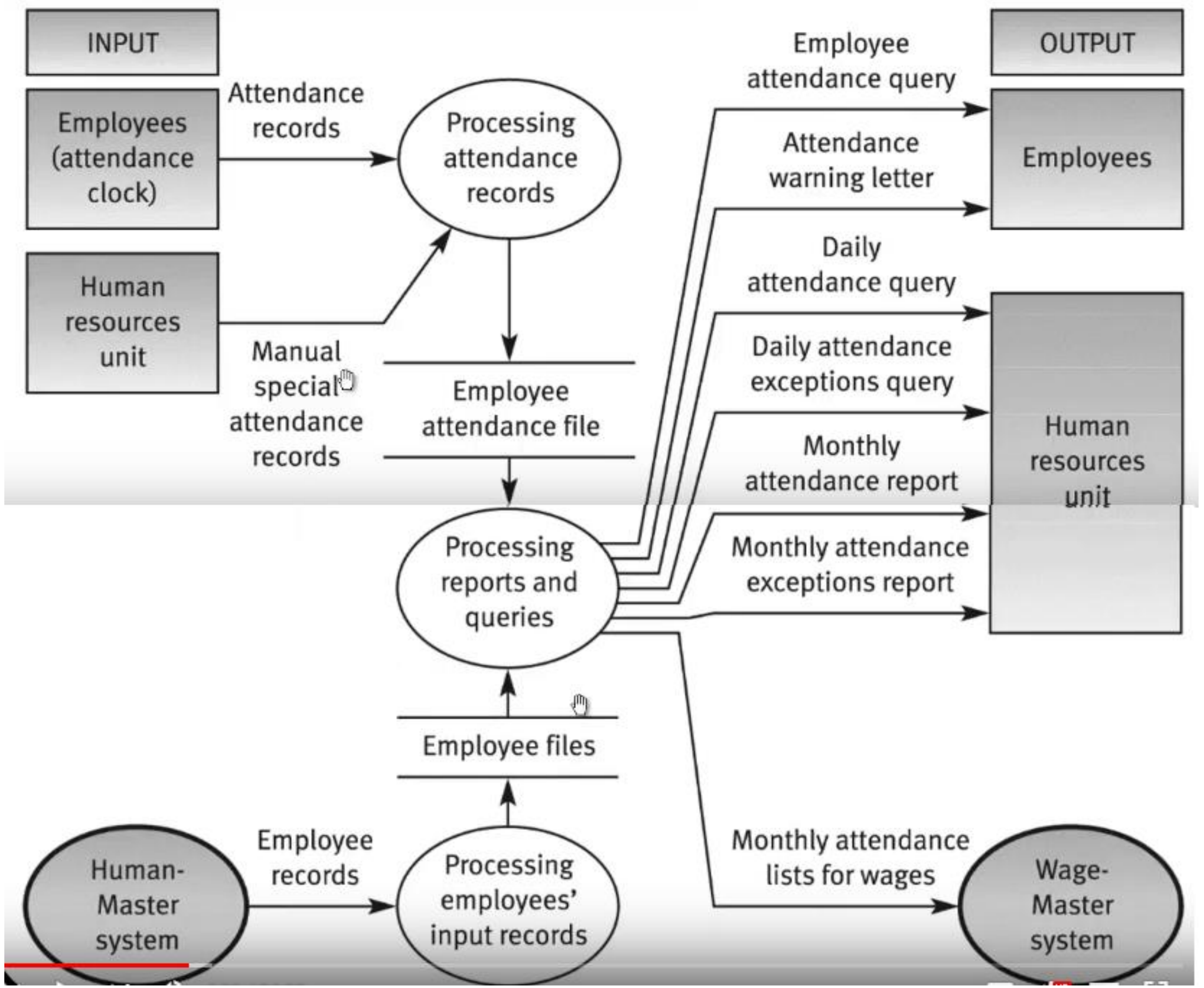
User wants to add, change and delete customer data, wants to inquire on Customer and also requires four different reports on Customer with calculated data.

User wants to add, change, delete Product data, wants to inquire on Product and also requires a report on Product with calculated data.

User wants to inquire on Supplier using supplier number and also requires a report on Supplier with totaling results.

All of these data are of average complexity and overall system is moderately complex i.e. assume processing complexity adjustment factor is 46.

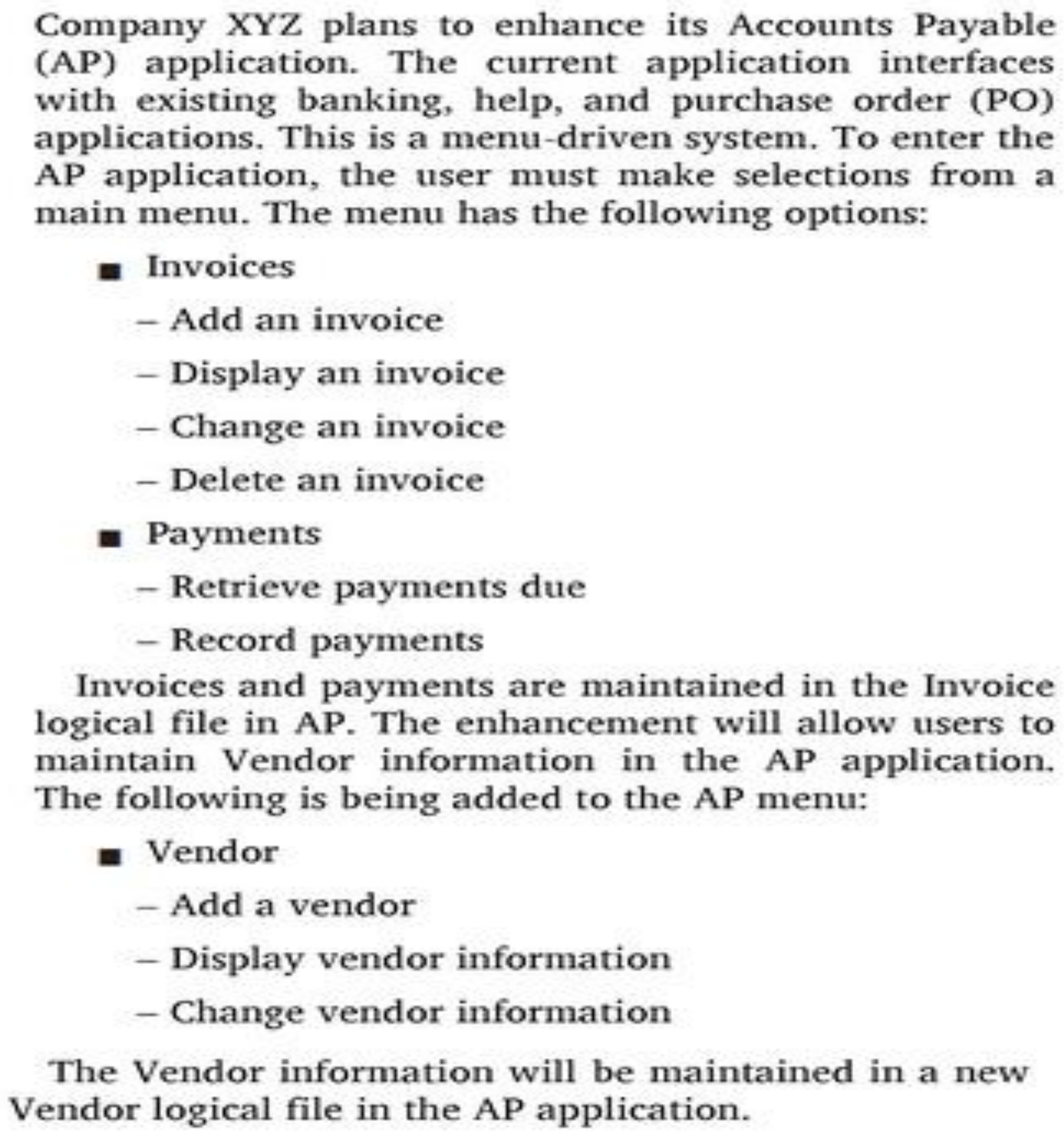
EXERCISE 2 EMPLOYEE TRACKING SYSTEM


INPUT
Employees (attendance clock)
Attendance records
Processing attendance records
Human resources unit
Manual special attendance records
Employee attendance file
Processing reports and queries
Employee files
Human-Master system
Employee records
Processing employees' input records
OUTPUT
Employee attendance query
Employees
Attendance warning letter
Daily attendance query
Daily attendance exceptions query
Monthly attendance report
Monthly attendance exceptions report
Human resources unit
Monthly attendance lists for wages
Wage-Master system

EXERCISE 3

A retail store has developed a new application, Frequent Buyer Program (FBF), to track customer purchases. The customer fills out a paper application and gives it to the store clerk. The clerk then adds the customer’s information online. The clerk can also list customers, view a customer’s detailed information and change a customer’s info. A report is produced daily listed customers that were added with their addresses.

EXERCISE 4



Company XYZ plans to enhance its Accounts Payable (AP) application. The current application interfaces with existing banking, help, and purchase order (PO) applications. This is a menu-driven system. To enter the AP application, the user must make selections from a main menu. The menu has the following options:

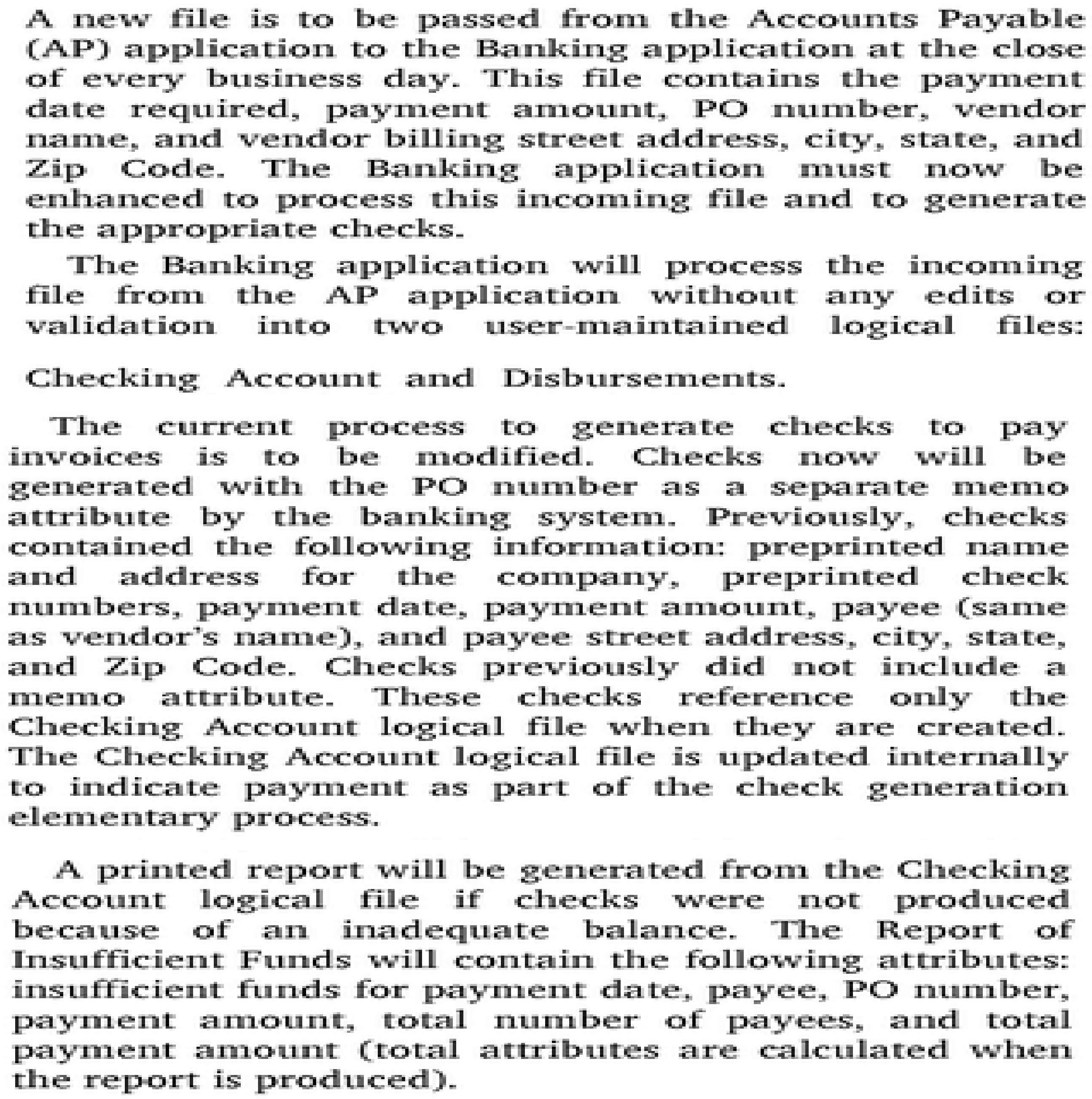
- Invoices
  - Add an invoice
  - Display an invoice
  - Change an invoice
  - Delete an invoice
- Payments
  - Retrieve payments due
  - Record payments

Invoices and payments are maintained in the Invoice logical file in AP. The enhancement will allow users to maintain Vendor information in the AP application. The following is being added to the AP menu:

- Vendor
  - Add a vendor
  - Display vendor information
  - Change vendor information

The Vendor information will be maintained in a new Vendor logical file in the AP application.

EXERCISE 5



A new file is to be passed from the Accounts Payable (AP) application to the Banking application at the close of every business day. This file contains the payment date required, payment amount, PO number, vendor name, and vendor billing street address, city, state, and Zip Code. The Banking application must now be enhanced to process this incoming file and to generate the appropriate checks.

The Banking application will process the incoming file from the AP application without any edits or validation into two user-maintained logical files:

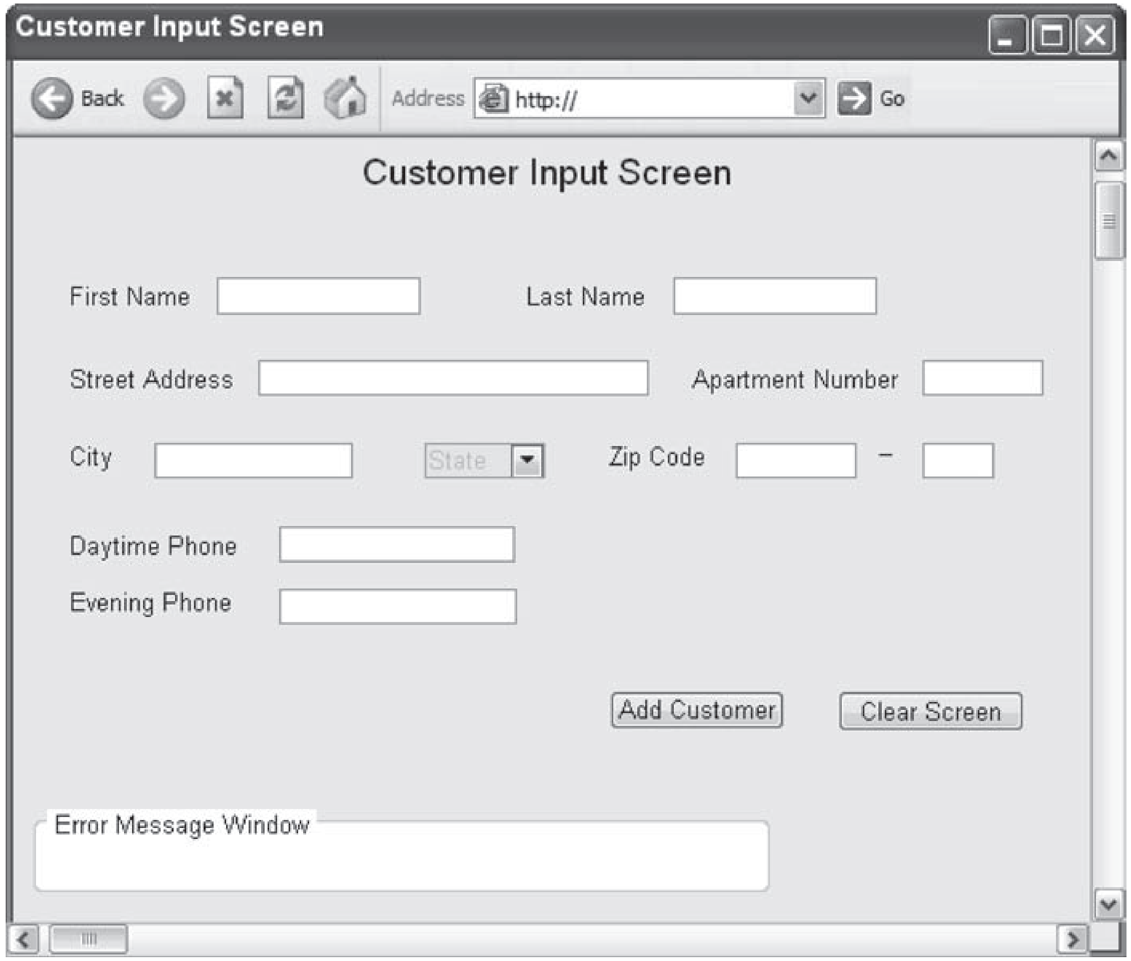
Checking Account and Disbursements.

The current process to generate checks to pay invoices is to be modified. Checks now will be generated with the PO number as a separate memo attribute by the banking system. Previously, checks contained the following information: preprinted name and address for the company, preprinted check numbers, payment date, payment amount, payee (same as vendor's name), and payee street address, city, state, and Zip Code. Checks previously did not include a memo attribute. These checks reference only the Checking Account logical file when they are created. The Checking Account logical file is updated internally to indicate payment as part of the check generation elementary process.

A printed report will be generated from the Checking Account logical file if checks were not produced because of an inadequate balance. The Report of Insufficient Funds will contain the following attributes: insufficient funds for payment date, payee, PO number, payment amount, total number of payees, and total payment amount (total attributes are calculated when the report is produced).

EXERCISE 6

An Application A is created where there are two transaction types: Add Employee (by entering Employee ID, Employee Name and Employee Hire Date) and Update Employee (Employee ID cannot be updated whereas other two fields can be updated). The Employee Job Assignment attribute on the input record is validated against the Job Assignment logical file maintained in Application C; this is the only attribute accessed in the Job Assignment logical file by Application A; however, the Job Assignment logical file has 51 attributes within Application C. If the validation passes, the Employee logical file is updated in Application A; a total of 12 fields are maintained and/or referenced in the Employee logical file within Application A. If the validation fails, no update is made and an error report is produced. Also, the same Application A, contains a screen related to customer management. Use the screen in below Figure as a reference. Note that the “Add Customer” button causes the Customer data maintained inside the application boundary to be updated with all the fields entered by the user (assume all fields are to be entered mandatorily, consider two fields of zip code as single one). The “Clear Screen” button causes all of the fields to be erased. The “Error Message Window” displays any errors associated with validations performed against an externally maintained Zip Code Data file after the “Add Customer” button is pressed. All the data is stored in the Customer file.



SYSTEM TEST AUTOMATION

Benefits of system test automation

- Test engineer productivity
- Coverage of regression testing
- Re-usability of test cases
- Consistency in testing
- Test interval reduction
- Reduces software maintenance cost
- Increased test effectiveness

EVALUATION AND SELECTION OF TEST TOOLS

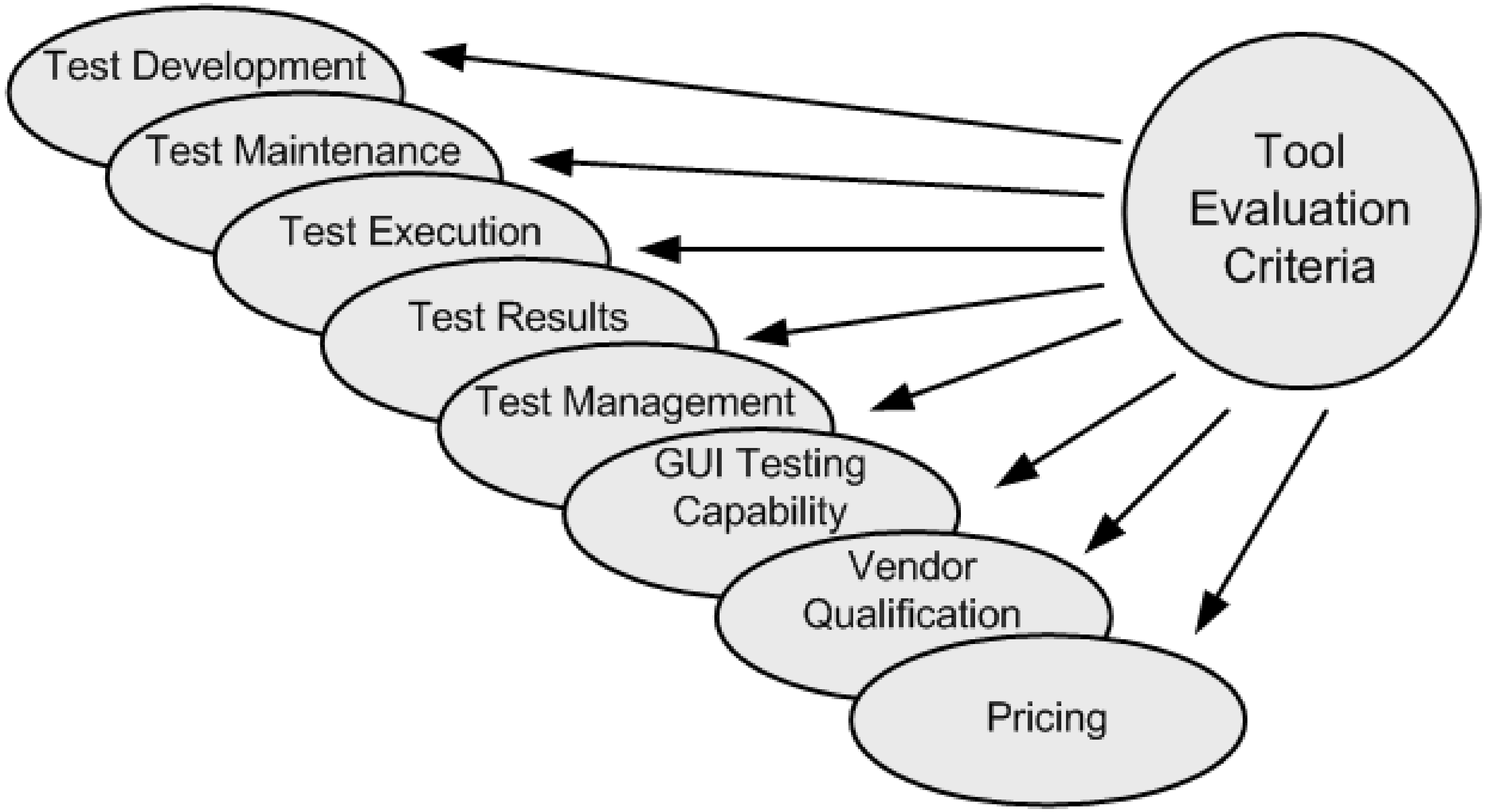


Figure 12.3: Broad criteria of test automation tool evaluation

CHARACTERISTICS OF AUTOMATION TEST CASES

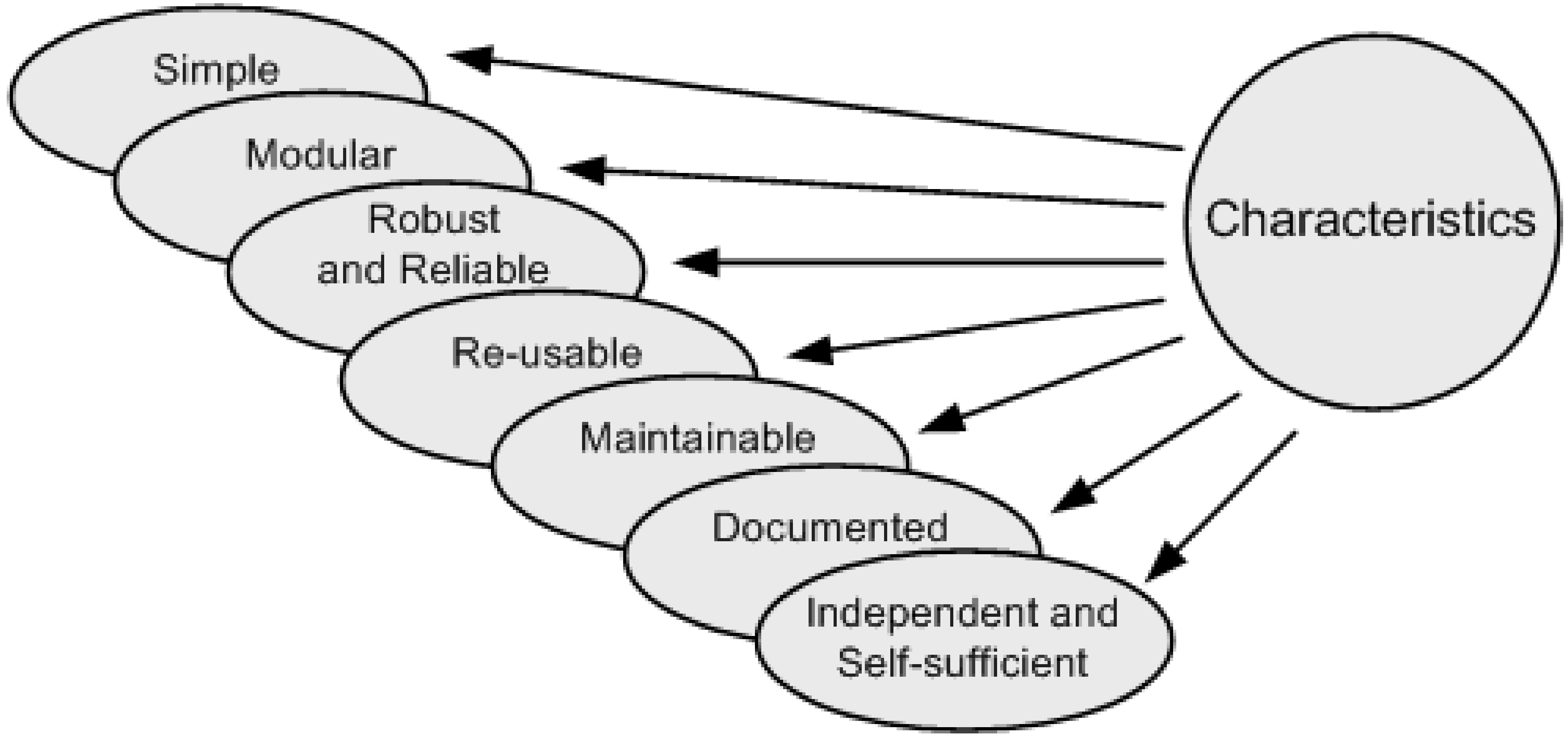


Figure 12.5: Characteristics of automated test cases

EARLY AUTOMATION

Advantages of early automation:

- (a) More time is available to evolve, fine-tune, and improve the automated test cases.
- (b) Testers learn the automated tool(s) and test environments via hands-on experience.
- (c) Early automation provides earlier feedback to the developers about the defects.

Disadvantages of early automation:

- (a) The development project needs the discipline of a visual freeze and stable functional specification.
- (b) Early automation can delay the first release ofthe system, because of unanticipated problems with automation and diversion of test resources from manual testing.

LATE AUTOMATION

Advantages of late automation:

- (a) The system under test is more stable.
- (b) There is less maintenance, i.e., throw-away and re-work of the automated test cases are minimal.
- (c) Testers have more time to learn the automation tools and processes before automating.
- (d) Manual testing provides direction for development of test cases.

Disadvantages of late automation:

- (a) Automation which is delayed may never be done because of pressure of other duties.
- (b) Testers do not benefit from an early intensive hands-on learning experience.
- (c) There is little or no regression testing of the early system release, because manual regression testing under deadline pressure is very difficult.

THANK YOU!!