BITS Pilani presentation

BITS Pilani
Pilani Campus

Dr. Nagesh BS

innovate achieve lead

BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Module 5

Test levels and types

Three levels of testing:

1. **Unit testing**
2. **Integration testing**
3. **System testing**

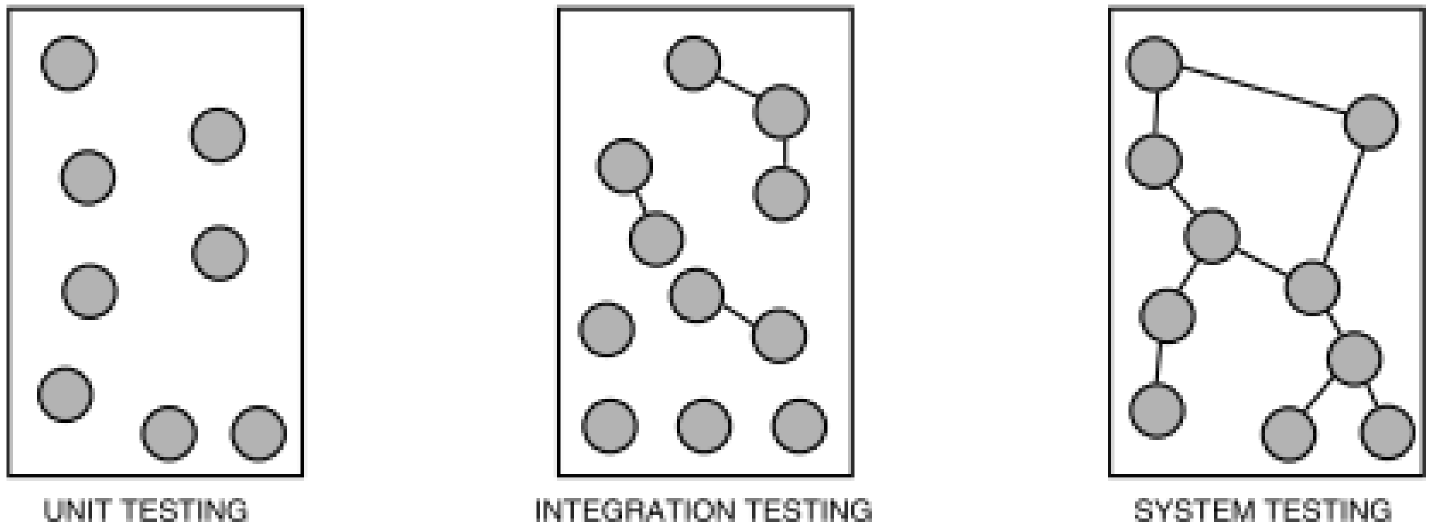
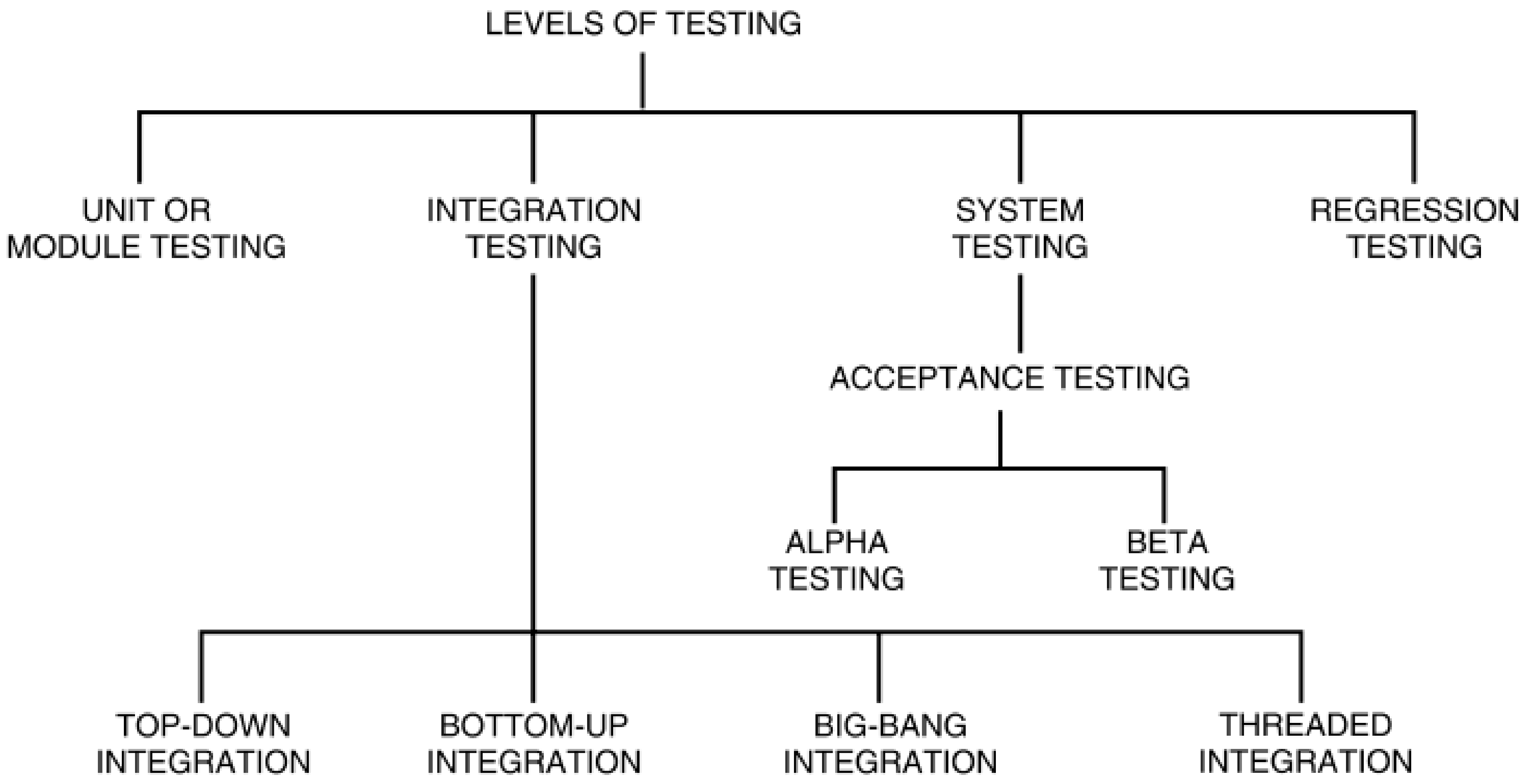


FIGURE 7.1 Levels of Testing.


LEVELS OF TESTING
UNIT OR MODULE TESTING
INTEGRATION TESTING
SYSTEM TESTING
REGRESSION TESTING
ACCEPTANCE TESTING
ALPHA TESTING
BETA TESTING
TOP-DOWN INTEGRATION
BOTTOM-UP INTEGRATION
BIG-BANG INTEGRATION
THREADED INTEGRATION

Unit (or module) testing

Unit (or module) testing "is the process of taking a module (an atomic unit) and running it in isolation from the rest of the software product by using prepared test cases and comparing the actual results with the results predicted by the specification and design module."

It is a white-box testing technique.

Importance of unit testing:

1. Because modules are being tested individually, testing becomes easier.
2. It is more exhaustive.
3. Interface errors are eliminated.

INTEGRATION TESTING

A system is composed of multiple components or modules that comprise hardware and software.

Integration is defined as the set of interactions among components.

Testing the interaction between the modules and interaction with other systems externally is called integration testing.

The architecture and design can give the details of interactions within systems.

Classification of integration testing

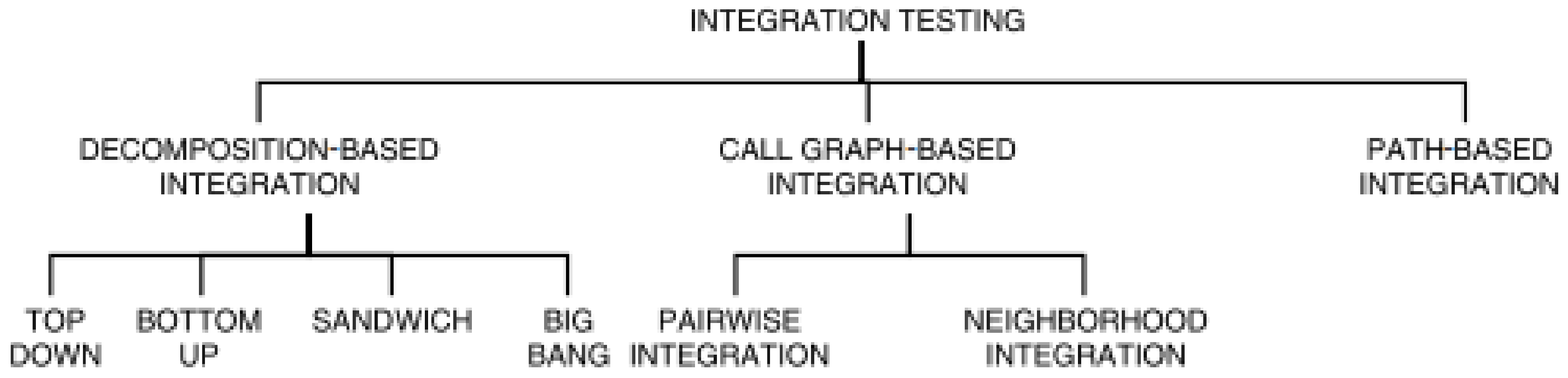


FIGURE 7.4

Decomposition-based Integration: *functional decomposition of the system to be tested which* is represented as a tree or in textual form.

- The goal of decomposition-based integration is to test the interfaces among separately tested units.

Types Of Decomposition-based Techniques :Top-down Integration Approach

It begins with the main program, i.e., the root of the tree.

Any lower-level unit that is called by the main program appears as a “stub.”

A stub is a piece of throw-away code that emulates a called unit.

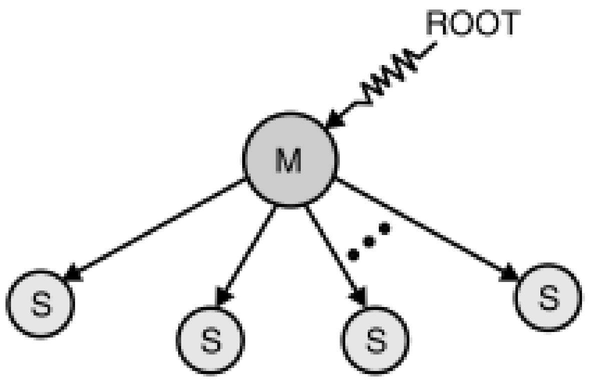


FIGURE 7.5 Stubs.

Bottom-up Integration Approach

It is a mirror image to the top-down order with the difference that stubs are replaced by driver modules that emulate units at the next level up in the tree.

start with the leaves of the decomposition tree and test them with specially coded drivers. Less throw-away code exists in drivers than there is in stubs.

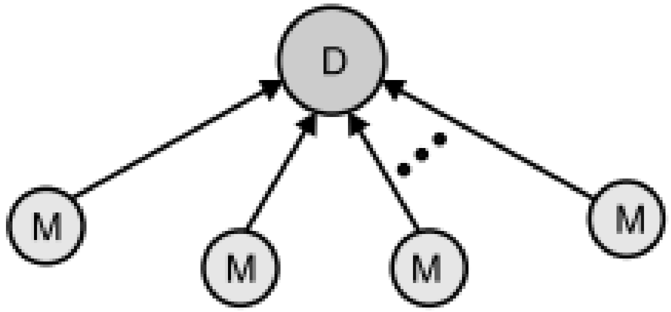


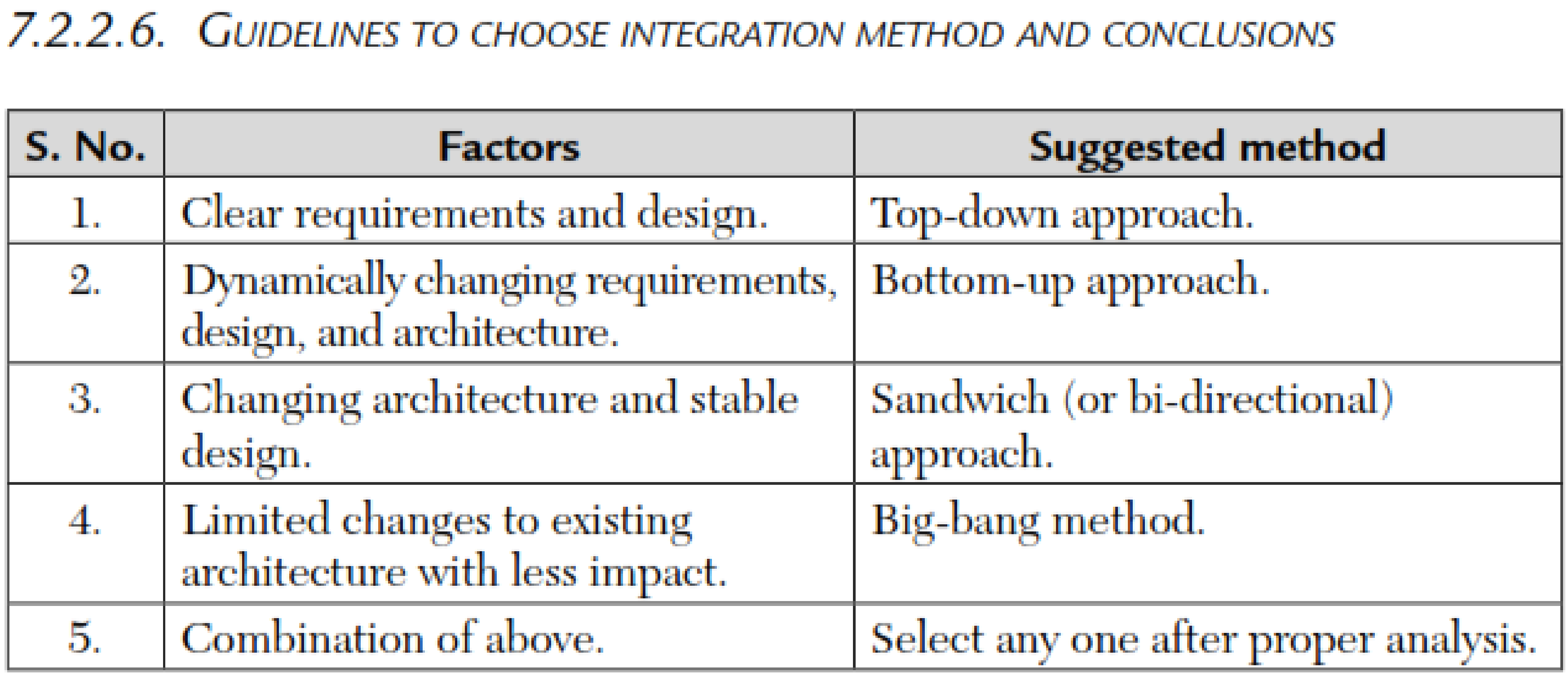
FIGURE 7.6 Drivers.

Sandwich Integration Approach

- It is a combination of top-down and bottom-up integration, bidirectional integration
- There will be less stub and driver development effort.
- Combination of the top-down and bottom-up integration approaches, it is also called bidirectional integration.
- It is performed initially with the use of stubs and drivers.
- Focuses on those components which need focus and are new.

Big-bang Integration

- Instead of integrating component by component and testing, this approach waits until all the components arrive and one round of integration testing is done. This is known as *big-bang integration.*
- *It reduces testing effort and* removes duplication in testing for the multi-step component integrations.
- Big-bang integration is ideal for a product where the interfaces are stable with fewer number of defects.



7.2.2.6. Guidelines to Choose Integration Method and Conclusions

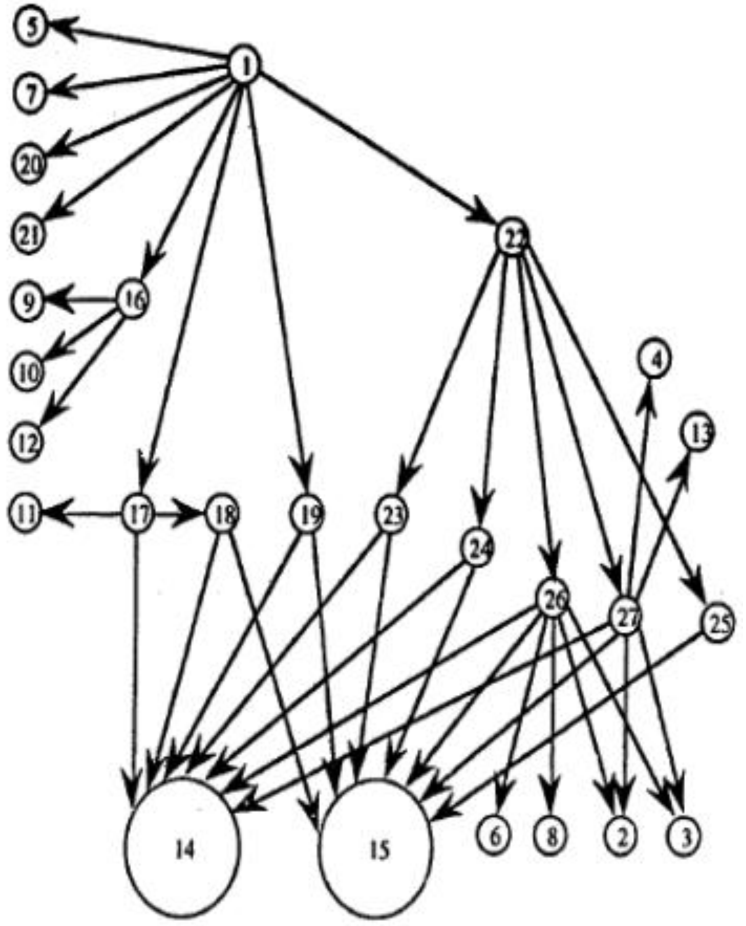
S. No.	Factors	Suggested method
1.	Clear requirements and design.	Top-down approach.
2.	Dynamically changing requirements, design, and architecture.	Bottom-up approach.
3.	Changing architecture and stable design.	Sandwich (or bi-directional) approach.
4.	Limited changes to existing architecture with less impact.	Big-bang method.
5.	Combination of above.	Select any one after proper analysis.

Call Graph-based Integration

- Structural testing
- Call graph is a directed graph .
- Call graph is a directed graph thus, we can use it as a program graph.
    - *Pairwise Integration*
    - *Neighborhood Integration*

Table 13.1 SATM Units and Abbreviated Names

Unit Number	*Level Number*	*Unit Name*
1	1	SATM System
A	1.1	Device Sense & Control
D	1.1.1	Door Sense & Control
2	1.1.1.1	Get Door Status
3	1.1.1.2	Control Door
4	1.1.1.3	Dispense Cash
E	1.1.2	Slot Sense & Control
5	1.1.2.1	WatchCardSlot
6	1.1.2.2	Get Deposit Slot Status
7	1.1.2.3	Control Card Roller
8	1.1.2.3	Control Envelope Roller
9	1.1.2.5	Read Card Strip
10	1.2	Central Bank Comm.
11	1.2.1	Get PIN for PAN
12	1.2.2	Get Account Status
13	1.2.3	Post Daily Transactions
B	1.3	Terminal Sense & Control
14	1.3.1	Screen Driver
15	1.3.2	Key Sensor
C	1.4	Manage Session
16	1.4.1	Validate Card
17	1.4.2	Validate PIN
18	1.4.2.1	GetPIN
F	1.4.3	Close Session
19	1.4.3.1	New Transaction Request
20	1.4.3.2	Print Receipt
21	1.4.3.3	Post Transaction Local
22	1.4.4	Manage Transaction
23	1.4.4.1	Get Transaction Type
24	1.4.4.2	Get Account Type
25	1.4.4.3	Report Balance
26	1.4.4.4	Process Deposit
27	1.4.4.5	Process Withdrawal



Pairwise Integration

- The main idea behind pairwise integration is to eliminate the stub/driver development effort.
- Instead of developing stubs and drivers, why not use the actual code?
- we restrict a session to only a pair of units in the call graph.
- The end result is that we have one integration test session for each edge in the call graph.
- This is not much of a reduction in sessions from either topdown or bottom-up but it is a drastic reduction in stub/driver development.

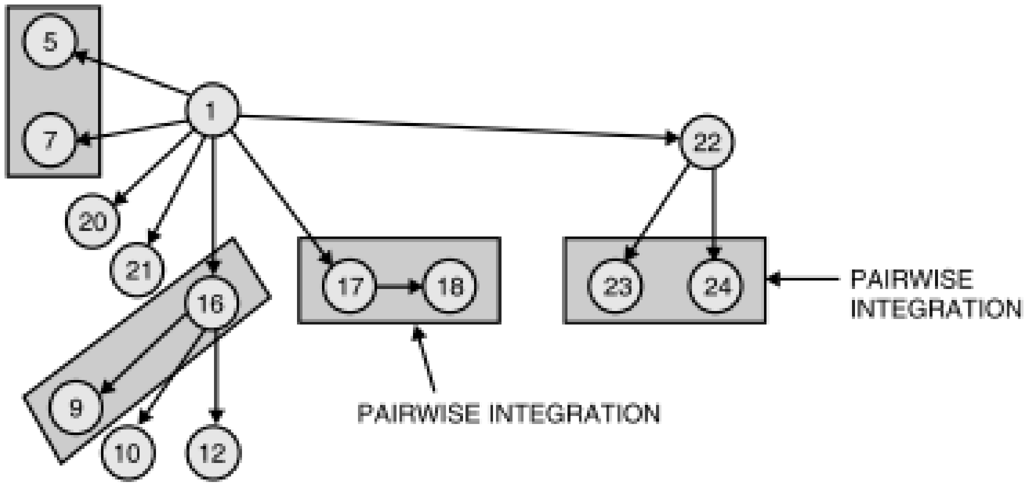


FIGURE 7.7 Pairwise Integration.

Neighborhood Integration

The neighborhood of a node in a graph is the set of nodes that are one edge away from the given node.

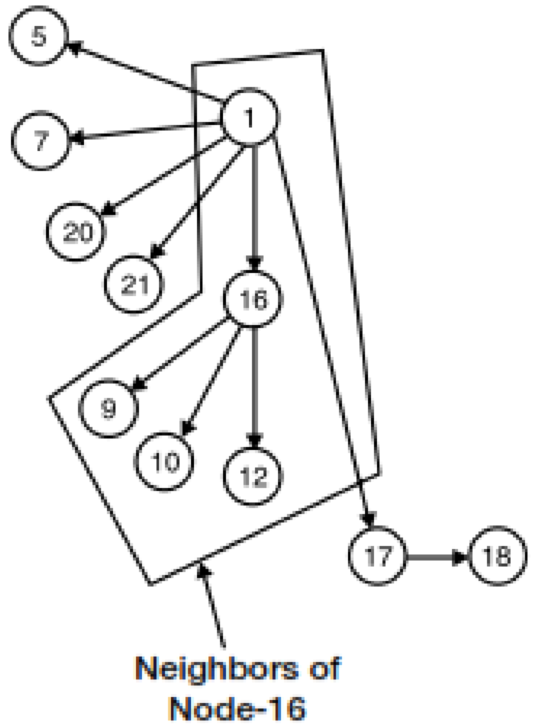


FIGURE 7.8 Neighborhood Integration.

Path-based Integration

- The combination of structural and functional testing is
- highly desirable at the unit level and it would be nice to have a similar capability for integration and system testing.
- "*Instead of testing interfaces among separately developed and tested units, we focus on interactions among these units.*"
- Here, *cofunctioning might be a* good term.
- Interfaces are structural whereas interaction is behavioral.

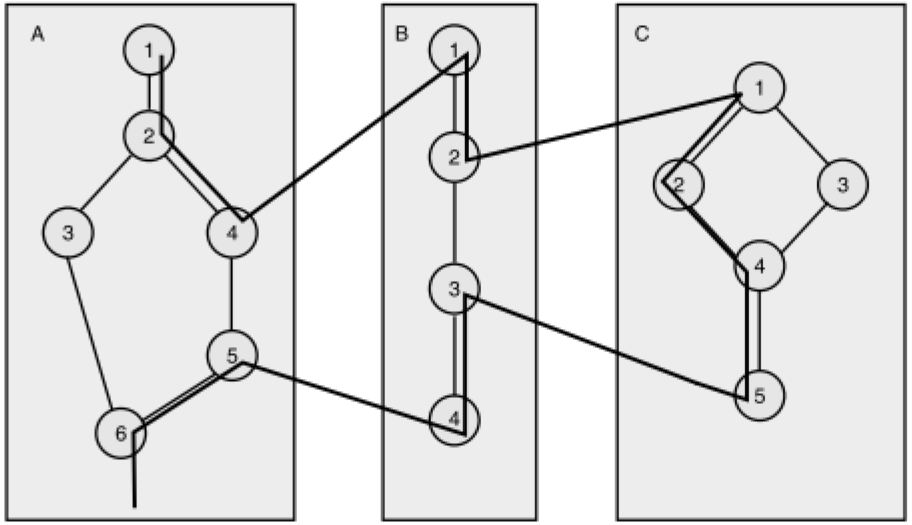


FIGURE 7.9 MM-Path Across Three Units (A, B, and C).

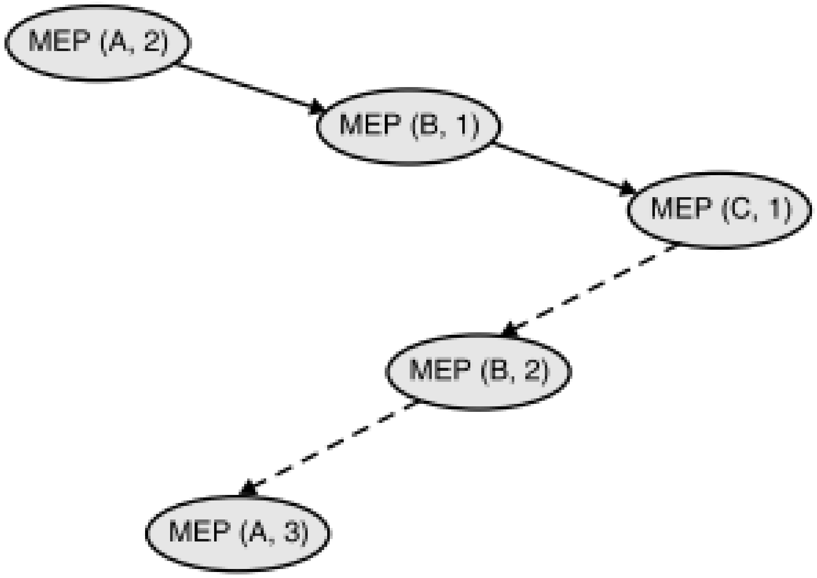


FIGURE 7.10 MM-Path Graph Derived from Figure 7.9.

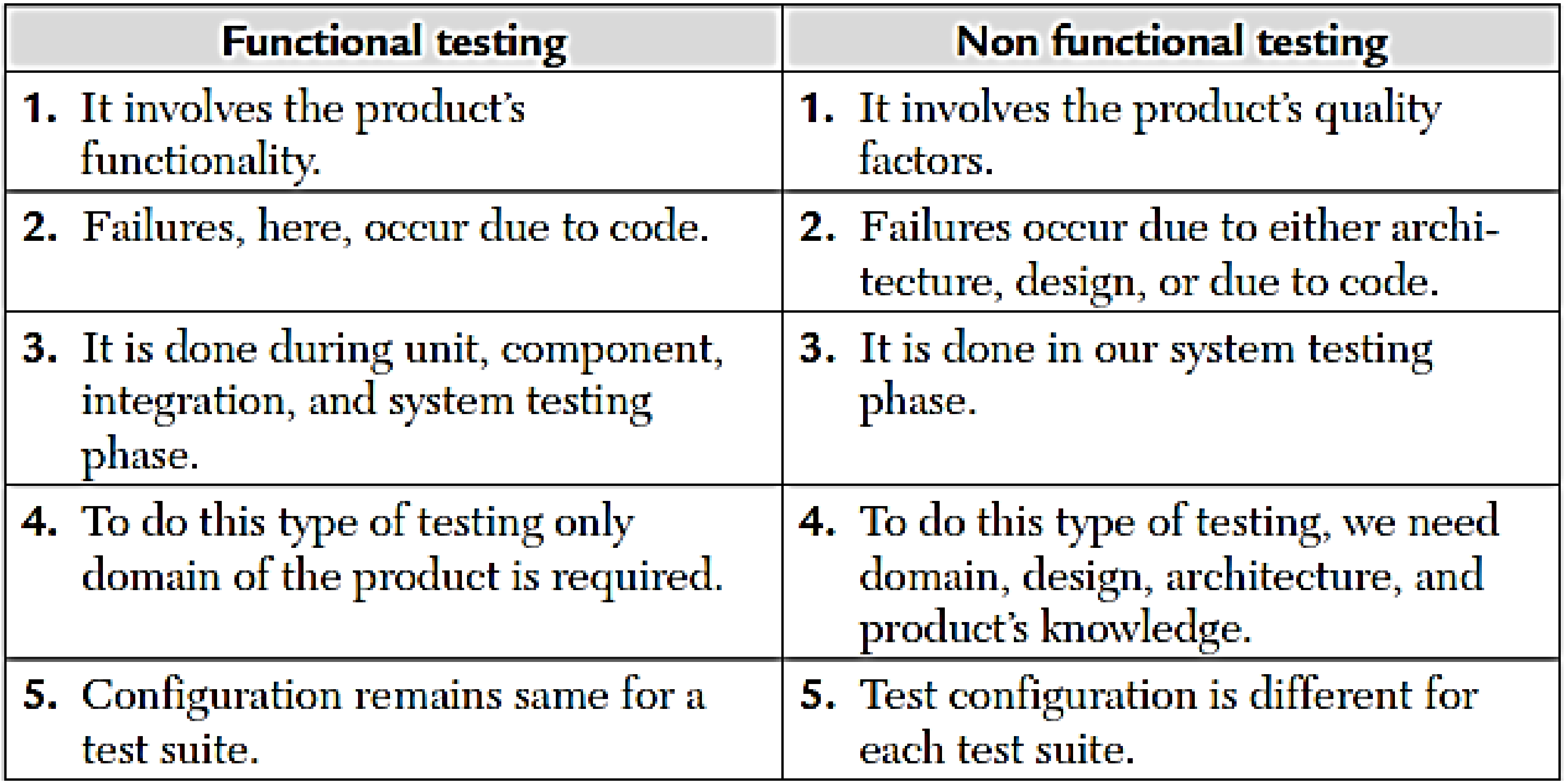
SYSTEM TESTING

The testing that is conducted on the complete integrated products and solutions to evaluate system compliance with specified requirements on functional and non functional aspects is called *system testing. It is done after* unit, component, and integration testing phases.

System testing is done to:

1. Provide independent perspective in testing as the team becomes more quality centric.
2. Bring in customer perspective in testing.
3. Provide a "fresh pair of eyes" to discover defects not found earlier by testing.
4. Test product behavior in a holistic, complete, and realistic environment.
5. Test both functional and non functional aspects of the product.
6. Build confidence in the product.
7. Analyze and reduce the risk of releasing the product.
8. Ensure all requirements are met and ready the product for acceptance testing.

- An independent test team normally does system testing.
- This independent test team is different from the team that does the component and integration testing.
- The system test team generally reports to a manager other than the product-manager to avoid conflicts and to provide freedom to individuals during system testing.
- Testing the product with an independent perspective and combining that with the perspective of the customer makes system testing unique, different, and effective.



Functional testing	Non functional testing
1. It involves the product's functionality.	1. It involves the product's quality factors.
2. Failures, here, occur due to code.	2. Failures occur due to either architecture, design, or due to code.
3. It is done during unit, component, integration, and system testing phase.	3. It is done in our system testing phase.
4. To do this type of testing only domain of the product is required.	4. To do this type of testing, we need domain, design, architecture, and product's knowledge.
5. Configuration remains same for a test suite.	5. Test configuration is different for each test suite.

Module 5

Test Execution Process

Test plan

The primary purpose of a test plan is to define the scope, approach, resources, and schedule for testing activities. It provides a systematic approach to ensure that the software meets specified requirements and quality standards.

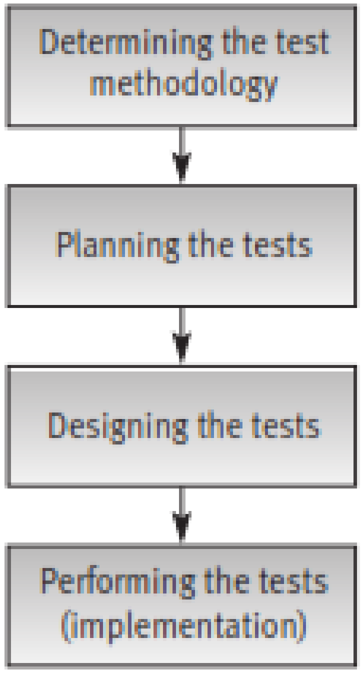
Key components of a test plan

- Objectives,
- Scope,
- Test Items,
- Test Environment,
- Testing Strategy,
- Test Schedule,
- Resource Requirements,
- Risk Management
- Test Deliverables.

The testing process

Planning, design and performance of testing are carried out throughout the software development process.

These activities are divided in phases, beginning in the design stage and ending when the software is installed at the customer's site.


Determining the test methodology
Planning the tests
Designing the tests
Performing the tests (implementation)

The testing process

Determining the test methodology phase

The main issues that testing methodology has to contend with are:

- The appropriate required software quality standard
- The software testing strategy.

Determining the appropriate software quality standard

The level of quality standard selected for a project depends mainly on the characteristics of the software's application.

Example 1: A software package for a hospital patient bed monitor requires the highest software quality standard considering the possibly severe consequences of software failure.

Determining the software testing strategy

The issues that have to be decided include:

- The testing strategy: should a big bang or incremental testing strategy beadopted? If incremental testing is preferable, should testing be performed bottom-up or top-down?
- Which parts of the testing plan should be performed according to the white box testing model?
- Which parts of the testing plan should be performed according to the automated testing model?

Planning the tests

The tests to be planned include:

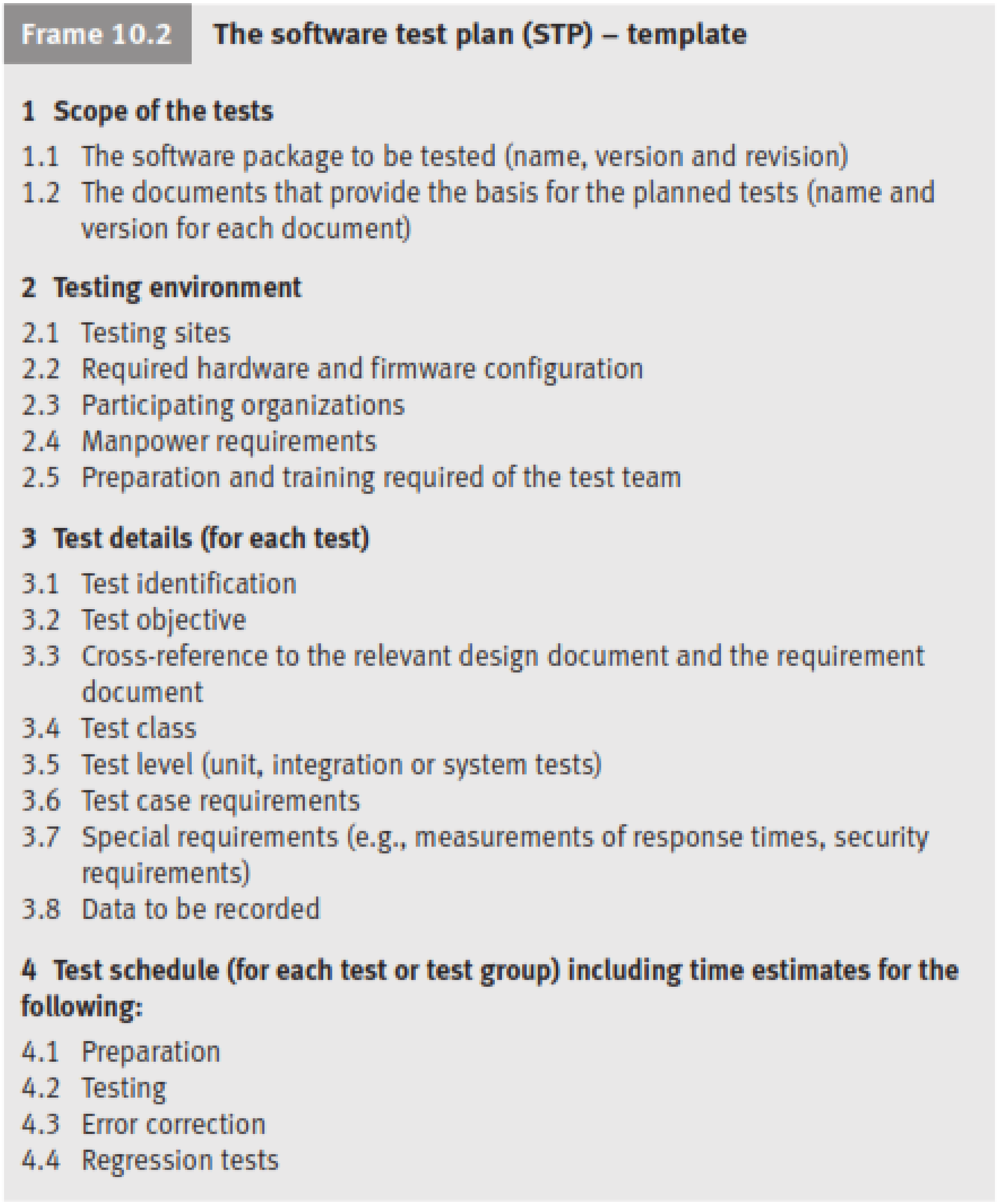
- Unit tests
- Integration tests
- System tests.

Consider the following issues before initiating a specific test plan:

- What to test?
- Which sources to use for test cases?
- Who is to perform the tests?
- Where to perform the tests?
- When to terminate the tests?

Test planning documentation

The planning stage of the software system tests is commonly documented in a “software test plan” (STP).



Frame 10.2 **The software test plan (STP) – template**

1 Scope of the tests

1.1 The software package to be tested (name, version and revision)
1.2 The documents that provide the basis for the planned tests (name and version for each document)

2 Testing environment

2.1 Testing sites
2.2 Required hardware and firmware configuration
2.3 Participating organizations
2.4 Manpower requirements
2.5 Preparation and training required of the test team

3 Test details (for each test)

3.1 Test identification
3.2 Test objective
3.3 Cross-reference to the relevant design document and the requirement document
3.4 Test class
3.5 Test level (unit, integration or system tests)
3.6 Test case requirements
3.7 Special requirements (e.g., measurements of response times, security requirements)
3.8 Data to be recorded

4 Test schedule (for each test or test group) including time estimates for the following:

4.1 Preparation
4.2 Testing
4.3 Error correction
4.4 Regression tests

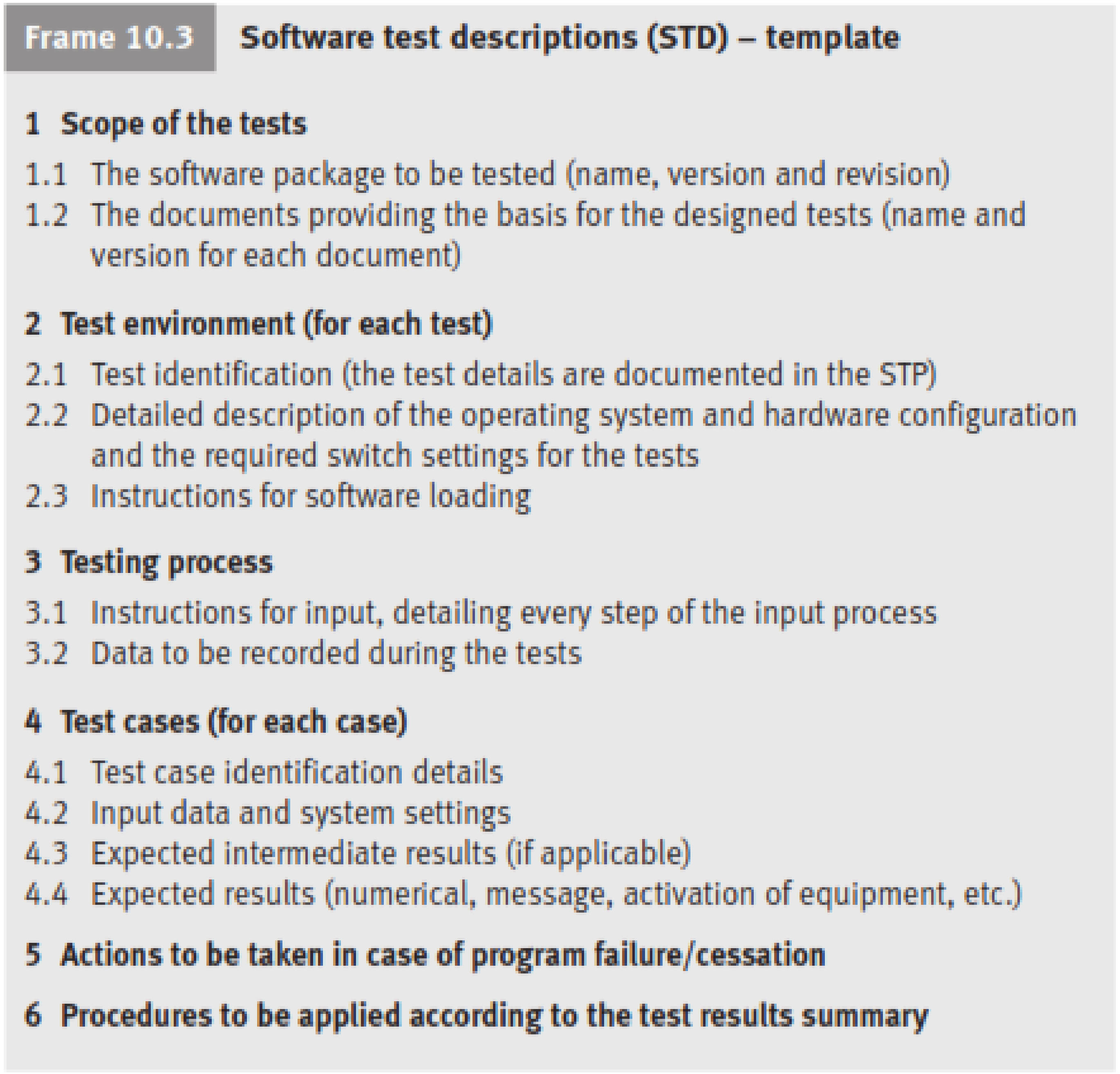
Test design

The products of the test design stage are:

- Detailed design and procedures for each test
- Test case database/file.

The test design is carried out on the basis of the software test plan as documented by STP.

The test procedures and the test case database/file may be documented in a “software test procedure” document and “test case file” document or in a single document called the “software test description” (STD).



Frame 10.3 **Software test descriptions (STD) – template**

1 Scope of the tests

1.1 The software package to be tested (name, version and revision)
1.2 The documents providing the basis for the designed tests (name and version for each document)

2 Test environment (for each test)

2.1 Test identification (the test details are documented in the STP)
2.2 Detailed description of the operating system and hardware configuration and the required switch settings for the tests
2.3 Instructions for software loading

3 Testing process

3.1 Instructions for input, detailing every step of the input process
3.2 Data to be recorded during the tests

4 Test cases (for each case)

4.1 Test case identification details
4.2 Input data and system settings
4.3 Expected intermediate results (if applicable)
4.4 Expected results (numerical, message, activation of equipment, etc.)

5 Actions to be taken in case of program failure/cessation

6 Procedures to be applied according to the test results summary

Test implementation

- The testing implementation phase activities consist of a series of tests, corrections of detected errors and re-tests (regression tests).
- Testing is culminated when the re-test results satisfy the developers.
- The tests are carried out by running the test cases according to the test procedures.
- Documentation of the test procedures and the test case database/file comprises the "software test description" (STD)
- Re-testing (also termed "regression testing") is conducted to verify that the errors detected in the previous test runs have been properly corrected, and that no new errors have entered as a result of faulty corrections.

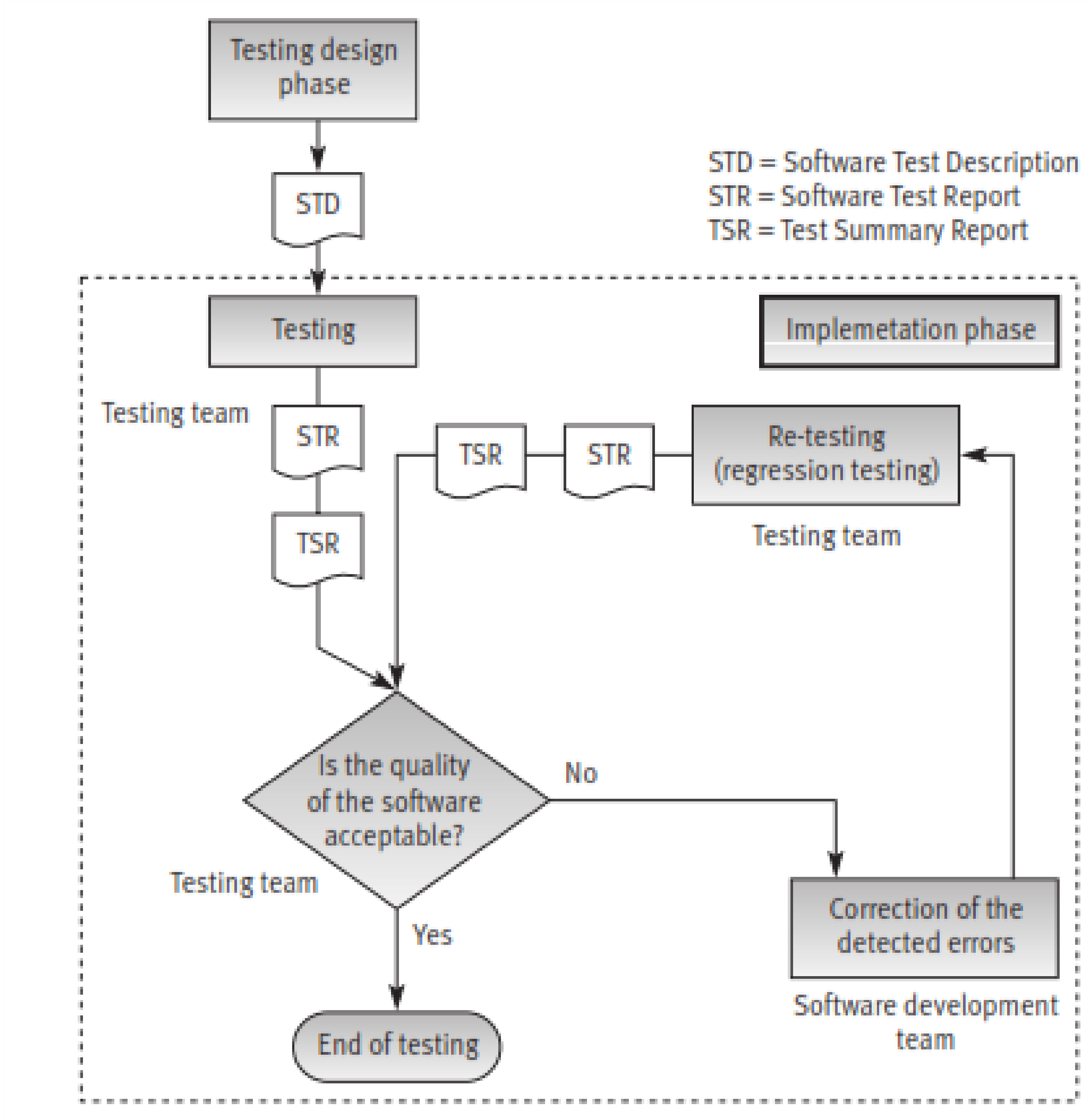
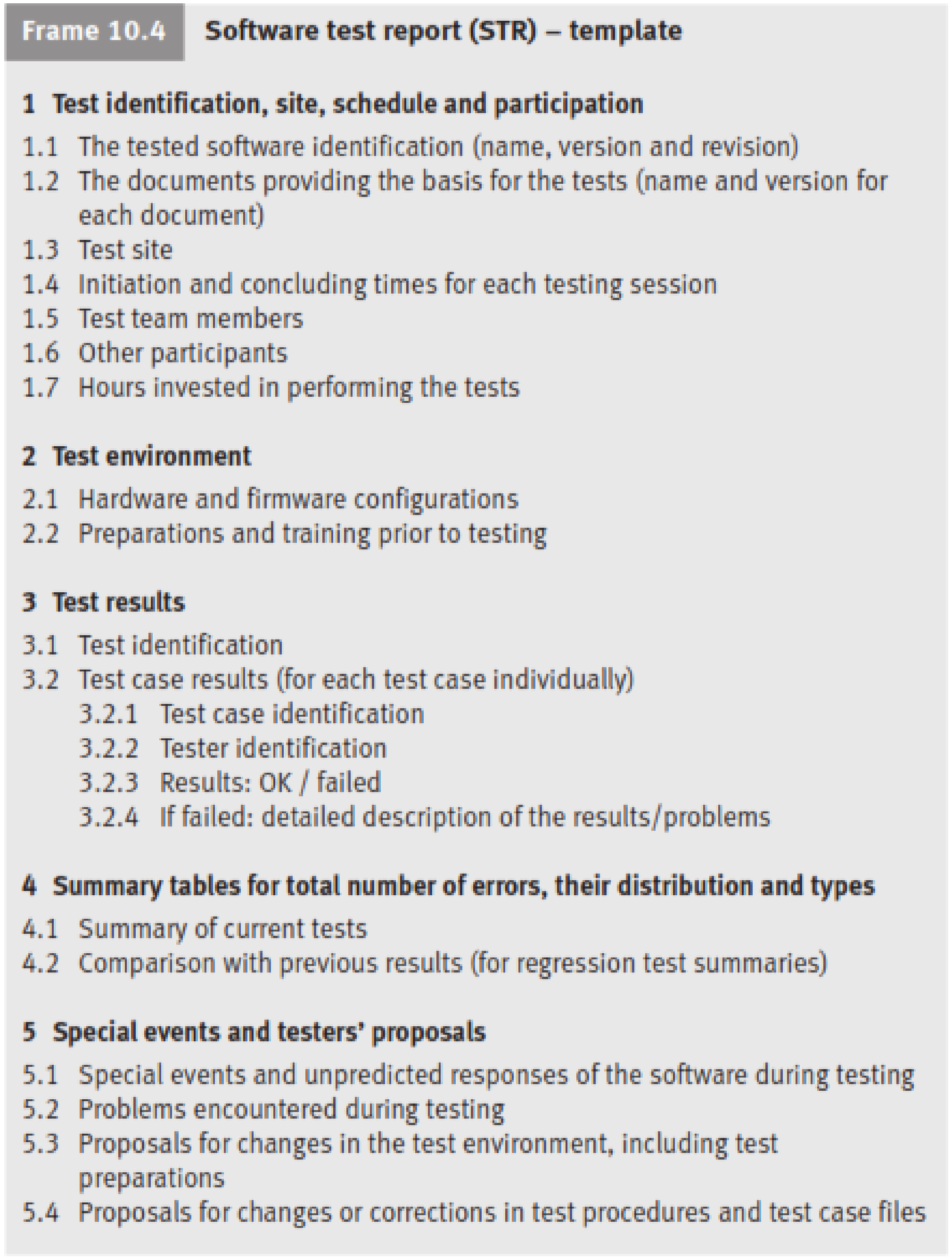

Testing design phase
STD
STD = Software Test Description
STR = Software Test Report
TSR = Test Summary Report
Testing
Implemetation phase
Testing team
STR
TSR
TSR
STR
Re-testing (regression testing)
Testing team
Is the quality of the software acceptable?
No
Testing team
Yes
Correction of the detected errors
Software development team
End of testing

Figure 10.2: Implementation phase activities

The summary of the set of tests planned for a software package (or software development project) is documented in the “test summary report” (TSR).



Frame 10.4 **Software test report (STR) – template**

1 Test identification, site, schedule and participation

1.1 The tested software identification (name, version and revision)
1.2 The documents providing the basis for the tests (name and version for each document)
1.3 Test site
1.4 Initiation and concluding times for each testing session
1.5 Test team members
1.6 Other participants
1.7 Hours invested in performing the tests

2 Test environment

2.1 Hardware and firmware configurations
2.2 Preparations and training prior to testing

3 Test results

3.1 Test identification
3.2 Test case results (for each test case individually)
 3.2.1 Test case identification
 3.2.2 Tester identification
 3.2.3 Results: OK / failed
 3.2.4 If failed: detailed description of the results/problems

4 Summary tables for total number of errors, their distribution and types

4.1 Summary of current tests
4.2 Comparison with previous results (for regression test summaries)

5 Special events and testers' proposals

5.1 Special events and unpredicted responses of the software during testing
5.2 Problems encountered during testing
5.3 Proposals for changes in the test environment, including test preparations
5.4 Proposals for changes or corrections in test procedures and test case files

Test case design

Test case data components

A test case is a documented set of the data inputs and operating conditions required to run a test item together with the expected results of the run.

The tester is expected to run the program for the test item according to the test case documentation, and then compare the actual results with the expected results noted in the documents.

If the obtained results completely agree with the expected results, no error is present or at least has been identified. When some or all of the results do not agree with the expected results, a potential error is identified.

Example

Consider the following test cases for the basic annual municipal property tax on apartments. The basic municipal property tax (before discounts to special groups of city dwellers) is based on the following parameters:

- *S*, the size of the apartment (in square yards)
- N, the number of persons living in the apartment
- A, B or C, the suburb's socio-economic classification.

The municipal property tax (MPT) is calculated as follows:

For class A suburbs: $MPT = (100 \times S) / (N + 8)$

For class B suburbs $MPT = (80 \times S) / (N + 8)$

For class C suburbs $MPT = (50 \times S) / (N + 8)$

The following are three test cases for the software module used to calculate the basic municipal property tax on apartments:

	Test case 1	Test case 2	Test case 3
Size of apartment – (square yards), *S*	250	180	98
Suburb class	A	B	C
No. of persons in the household, *N*	2	4	6
Expected result: municipal property tax (MPT)	**$2500**	**$1200**	**$350**

Automated testing

- Automated testing represents an additional step in the integration of computerized tools into the process of software development.
- These tools have joined computer aided software engineering (CASE) tools in performing a growing share of software analysis and design tasks.

Factors have motivated the development of automated testing tools:

Anticipated cost savings, shortened test duration, heightened thoroughness of the tests performed, improvement of test accuracy, improvement of result reporting as well as statistical processing andsubsequent reporting.

The process of automated testing

Automated software testing requires test planning, test design, test case preparation, test performance, test log and report preparation, re-testing after correction of detected errors (regression tests), and final test log and Report preparation including comparison reports.

The last two activities may be repeated several times.

Types of automated tests

Code auditing : The computerized code auditor checks the compliance of code to specified standards and procedures of coding. The auditor's report includes a list of the deviations from the standards and a statistical summary of the findings.

Coverage monitoring: Coverage monitors produce reports about the line coverage achieved when implementing a given test case file. The monitor's output includes the percentage of lines covered by the test cases as well as listings of uncovered lines.

These features make coverage monitoring a vital tool for white-box tests.

Functional tests : Automated functional tests often replace manual black-box correctness tests.

Prior to performance of these tests, the test cases are recorded into the test case database.

The tests are then carried out by executing the test cases through the test program.

The test results documentation includes listings of the errors identified in addition to a variety of summaries and statistics as demanded by the testers' specifications.

*Load tests :*The history of software system development contains many sad chapters of systems that succeeded in correctness tests but severely failed – and caused enormous damage – once they were required to operate under standard full load.

The damage in many cases was extremely high because the failure occurred “unexpectedly”, when the systems were supposed to start providing their regular software services.

*Test management :*Testing involves many participants occupied in actually carrying out the tests and correcting the detected errors.

Testing typically monitors performance of every item on long lists of test case files.

This workload makes timetable follow-up important to management. Computerized test management supports these and other testing management goals.

Advantages of automated tests

(1) Accuracy and completeness of performance.

(2) Accuracy of results log and summary reports.

(3) Comprehensiveness of information.

(4) Few manpower resources required to perform tests.

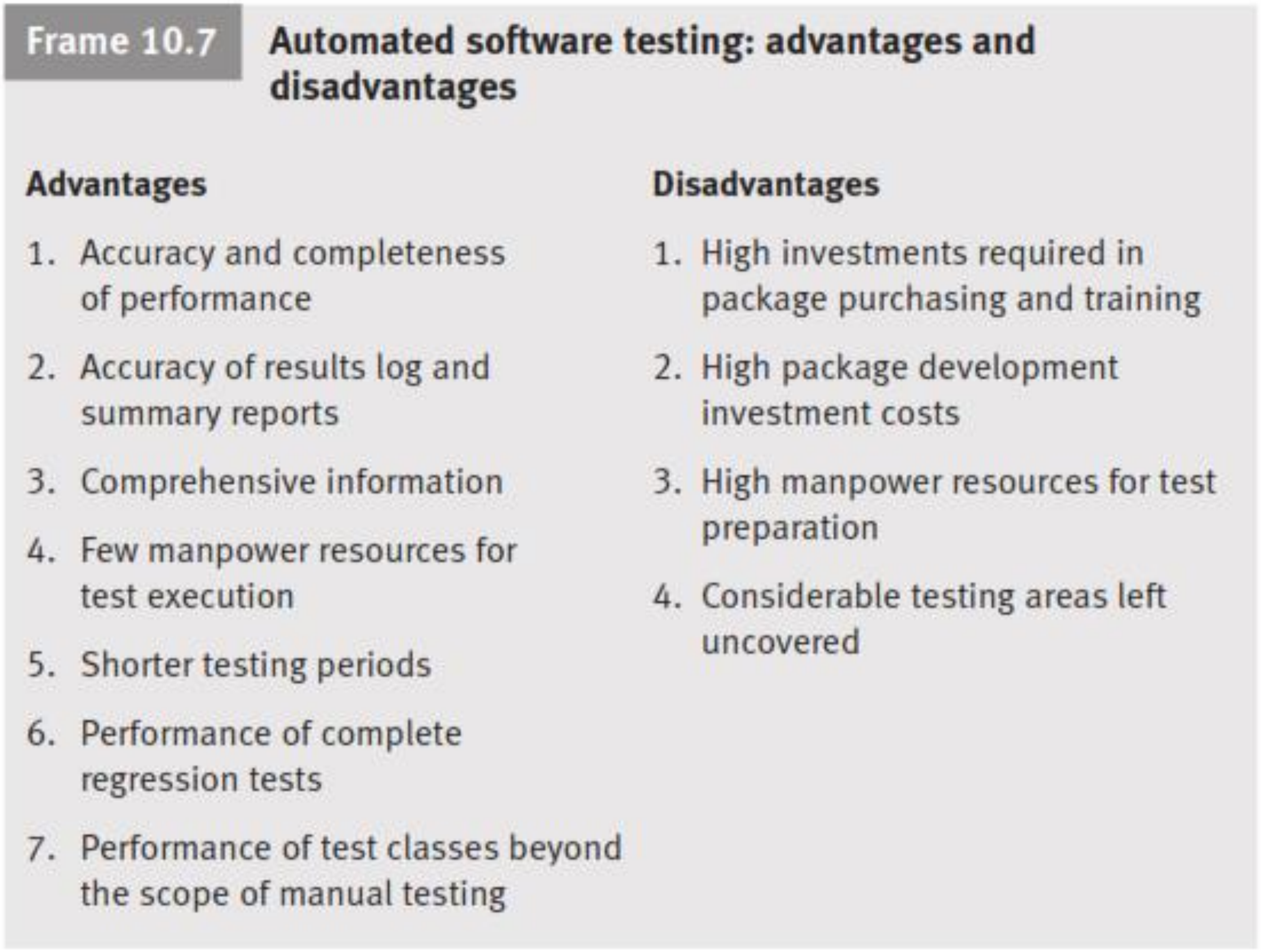
(5) Shorter duration of testing.

(6) Performance of complete regression tests.

(7) Performance of test classes beyond the scope of manual testing.

Disadvantages of automated testing

1) **High investments required in package purchasing and training.**
2) **High package development investment costs.**
3) **High manpower requirements for test preparation.**
4) **Considerable testing areas left uncovered.**



Frame 10.7 **Automated software testing: advantages and disadvantages**

Advantages

1. Accuracy and completeness of performance
2. Accuracy of results log and summary reports
3. Comprehensive information
4. Few manpower resources for test execution
5. Shorter testing periods
6. Performance of complete regression tests
7. Performance of test classes beyond the scope of manual testing

Disadvantages

1. High investments required in package purchasing and training
2. High package development investment costs
3. High manpower resources for test preparation
4. Considerable testing areas left uncovered

Alpha and beta site testing programs

- Alpha site and beta site tests are employed to obtain comments about quality from the package’s potential users.
- They are additional commonly used tools to identify software design and code errors in software packages in commercial over-the-counter sale (COTS).

Alpha site tests :“Alpha site tests” are tests of a new software package that are performed at the developer’s site.

Beta site tests : Once an advanced version of the software package is available, the developer offers it free of charge to one or more potential users.

The users install the package in their sites (usually called the “beta sites”), with the understanding that they will inform the developer of all the errors revealed during trials or regular usage.

REGRESSION TESTING TECHNIQUE

- Regression testing is used to confirm that fixed bugs have been fixed and that new bugs have not been introduced.
- How many cycles of regression testing are required will depend upon the project size. Cycles of regression testing may be performed once per milestone or once per build. Regression tests can be automated.

The Regression–Test Process

The regression-test process is shown in Figure 6.6.

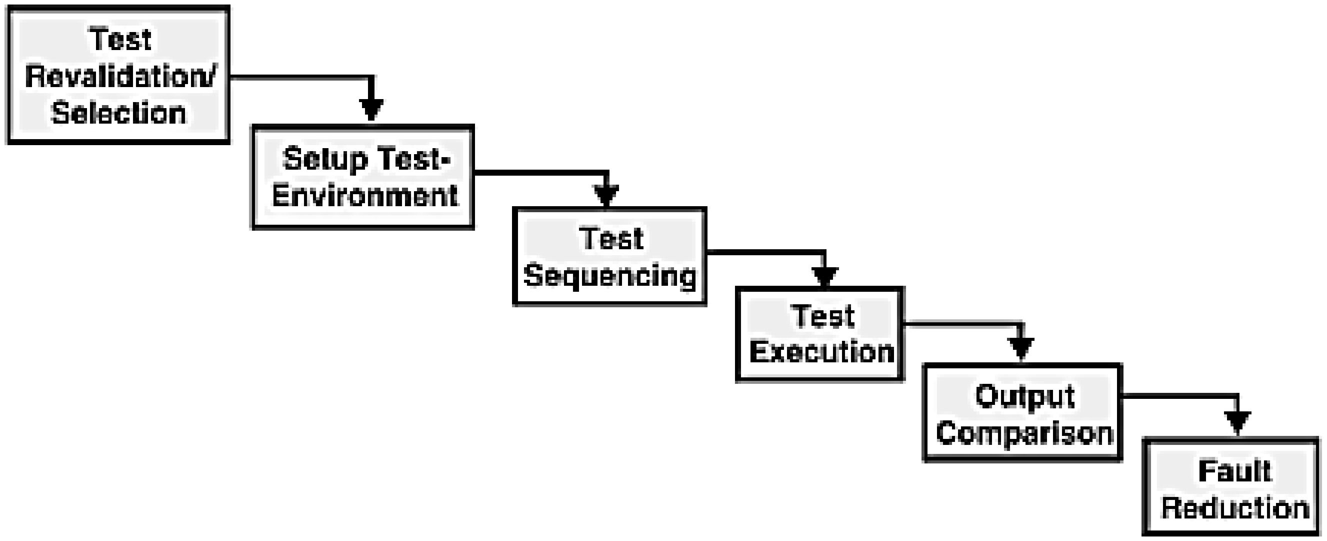


FIGURE 6.6

This process assumes that P′ (modified is program) available for regression testing. There is a long series of tasks that lead to P′ from P.

THANK YOU