



# BITS Pilani presentation

**BITS Pilani**  
Pilani Campus

Dr. Nagesh BS



**BITS Pilani**  
Pilani Campus



# **SE ZG501**

# **Software Quality Assurance and Testing**

## **Session 6**

# 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 testing team 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 <u>domain of the product</u> 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.

---

# 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

---

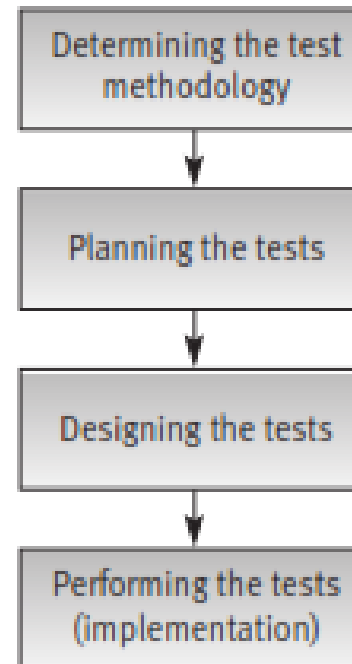


- Objectives,
- Scope,
- Test Items,
- Test Environment,
- Testing Strategy,
- Test Schedule,
- Resource Requirements,
- Risk Management
- Test Deliverables.

# 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 phase



The key challenges in testing methodology include :

- 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*

Key decisions to be made include:

- Selecting a testing approach: **Big Bang vs. Incremental**. If incremental, should it follow a **bottom-up or top-down** approach?
- Identifying which parts of the test plan should follow the **white-box testing** model.
- Determining which test cases should be executed using **automated testing**?

# 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.

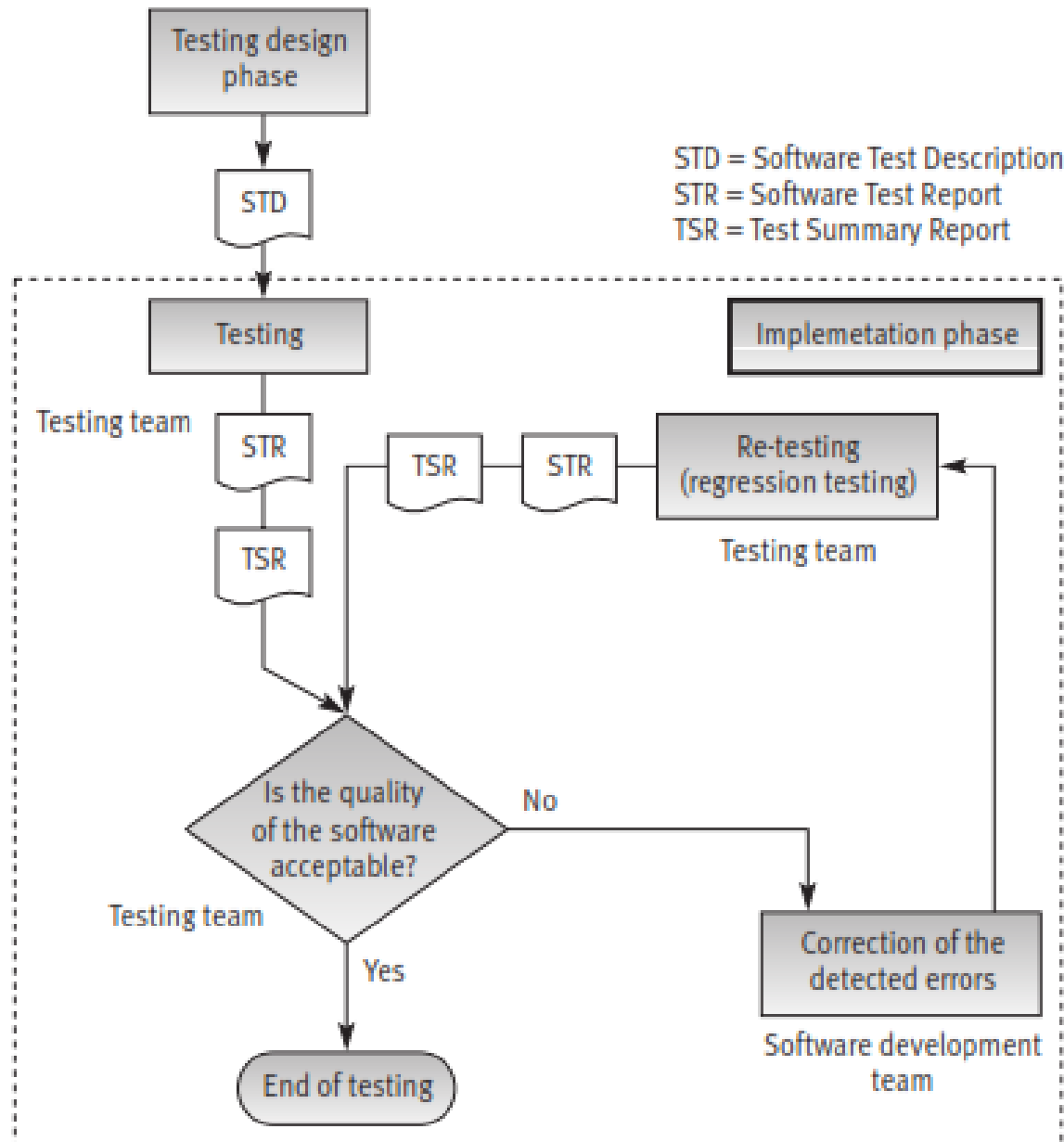


Figure 10.2: Implementation phase activities

---

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



**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
<b>Expected result:</b> <b>municipal property tax (MPT)</b>	<b>\$2500</b>	<b>\$1200</b>	<b>\$350</b>

# 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 and subsequent 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

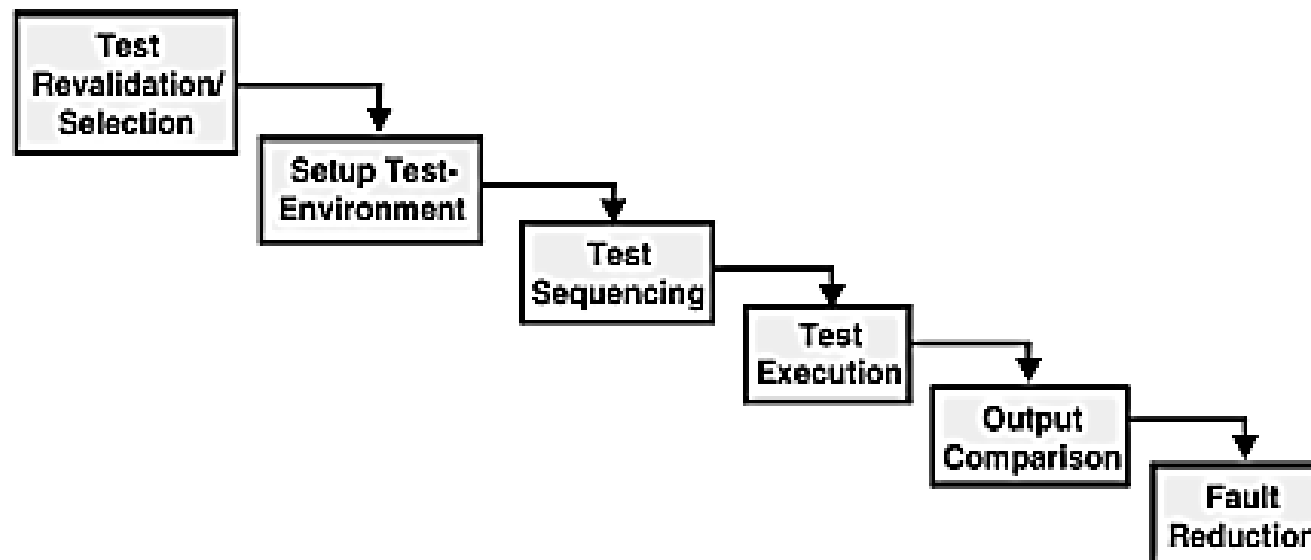
---



- Regression testing is used to confirm that **previously fixed bugs remain resolved** and that **no new bugs have 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  $P$ .

# Quality Audits and Project Assessments



- Humphrey (2005) [HUM 05] : years of data from thousands of software engineers showing that they unintentionally inject **100 defects per thousand lines of code**.
- He also indicates that **commercial software** typically includes from **one to ten errors per thousand lines of code** [HUM 02].
- These errors are like **hidden time bombs** that will explode when certain conditions are met.
- **Put practices in place to identify and correct these errors at each stage of the development and maintenance cycle.**

## Cost of quality

Quality costs = Prevention costs  
+ Appraisal or evaluation costs  
+ Internal and external failure costs  
+ Warranty claims and loss of reputation costs

The detection cost is the cost of verification or evaluation of a product or service during the various stages of the development process.

- One of the detection techniques is conducting **reviews**.
- Another technique is conducting **tests**.

- Quality of a software product begins in the first stage of the development process (defining requirements and specifications).
- **Reviews** will detect and correct errors in the early phase of development
- **Tests** will only be used when the code is available.
- we should not wait for the testing phase to begin to look for errors.
- Reviews range from informal to formal



# Informal reviews

An **informal review** lacks structure and standardization, leading to inconsistencies. Key issues include:

- **No documented process** – Different people conduct reviews in different ways.
- **Undefined roles** – No clear responsibilities for participants.
- **No specific objective** – Reviews don't focus on detecting faults.
- **Unplanned** – Conducted without proper scheduling.
- **No defect tracking** – Defects are not measured or recorded.
- **No management oversight** – Effectiveness is not evaluated.
- **No standards or checklists** – No guidelines to follow for identifying defects.

## Review

A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.

ISO 24765 [ISO 17a]

A process or meeting during which a software product, set of software products, or a software process is presented to project personnel, managers, users, customers, user representatives, auditors, or other interested parties for examination, comment, or approval.

IEEE 1028 [IEE 08b]

# IEEE 1028 Standard



## Types of Reviews in Software Engineering:

### 1. Walk-through & Inspection:

1. Formal review methods to identify defects and improve quality.

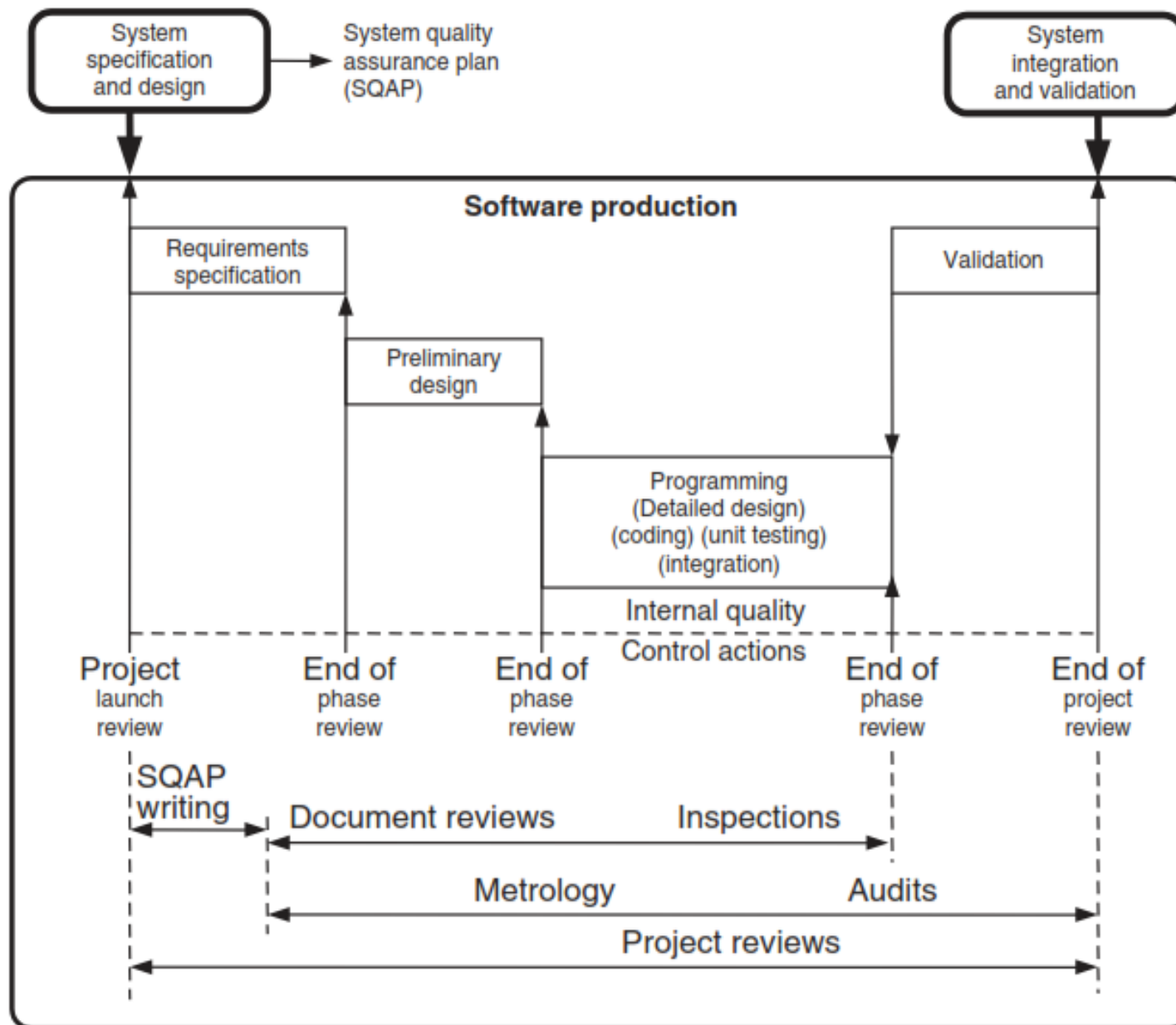
### 2. Personal Review & Desk Check:

1. Individual verification of code, documents, or design before peer review.

### 3. Peer Reviews:

1. Conducted by colleagues during development, maintenance, or operations.
2. Aims to identify errors, present alternatives, and discuss solutions.

 **Purpose:** Improve software quality by detecting defects early and enhancing collaboration.

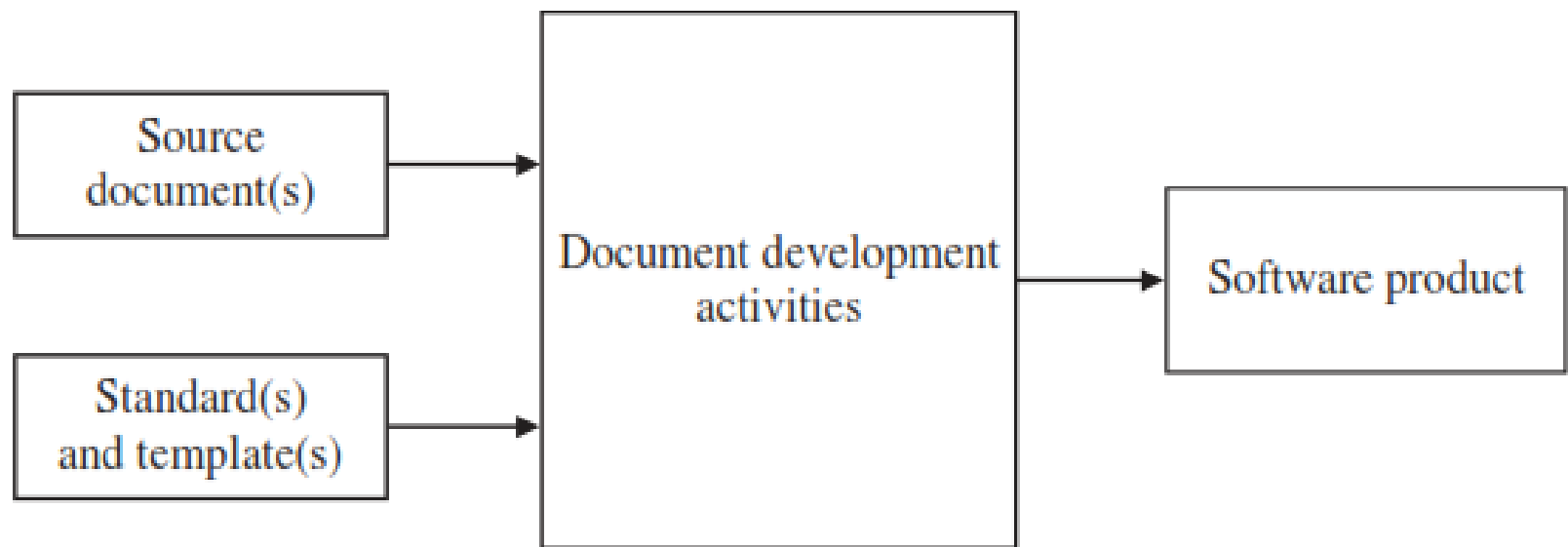


**Figure 5.1** Types of reviews used during the software development cycle [CEG 90] (© 1990 – ALSTOM Transport SA).

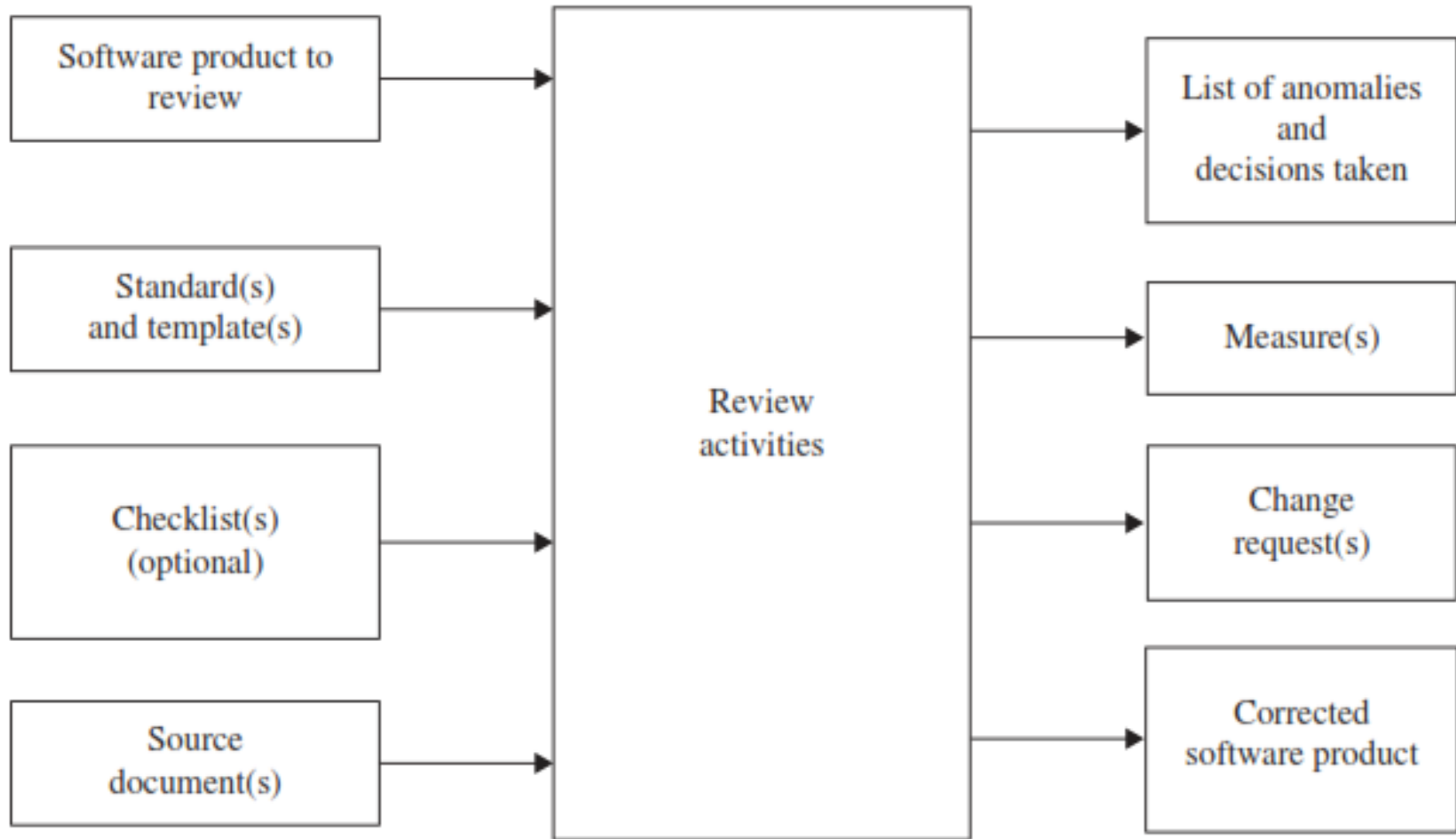
- Identify defects
- Assess / measure the quality of a document (e.g., the number of defects per page)
- Reduce the number of defects by correcting the defects identified
- Reduce the cost of preparing future documents (i.e., by learning the type of defects each developer makes, it is possible to reduce the number of defects injected in a new document)
- Estimate the effectiveness of a process (e.g., the percentage of fault detection)
- Estimate the efficiency of a process (e.g., the cost of detection or correction of a defect)
- Estimate the number of residual defects (i.e., defects not detected when software is delivered to customer)
- Reduce the cost of tests
- Reduce delays in delivery
- Determine the criteria for triggering a process
- Determine the completion criteria of a process
- Estimate the impacts (e.g., cost) of continuing with current plans, e.g. cost of delay, recovery, maintenance, or fault remediation
- Estimate the productivity and quality of organizations, teams and individuals
- Teach personnel to follow the standards and use templates
- Teach personnel how to follow technical standards
- Motivate personnel to use the organization's documentation standards
- Prompt a group to take responsibility for decisions
- Stimulate creativity and the contribution of the best ideas with reviews
- Provide rapid feedback before investing too much time and effort in certain activities
- Discuss alternatives
- Propose solutions, improvements
- Train staff
- Transfer knowledge (e.g., from a senior developer to a junior)
- Present and discuss the progress of a project
- Identify differences in specifications and standards
- Provide management with confirmation of the technical state of the project
- Determine the status of plans and schedules
- Confirm requirements and their assignment in the system to be developed

**Figure 5.2** Objectives of a review.

Source: Adapted from Gilb (2008) [Gilb, 2008] and IEEE 1028-IEEE 9811



**Figure 5.3** Process of developing a document.



**Figure 5.4** Review process.

# PERSONAL REVIEW AND DESK-CHECK REVIEW

---



- Inexpensive and very easy to perform.
- Personal reviews do not require the participation of additional reviewers,
- Desk-check reviews require at least one other person to review the work of the developer of a software product.

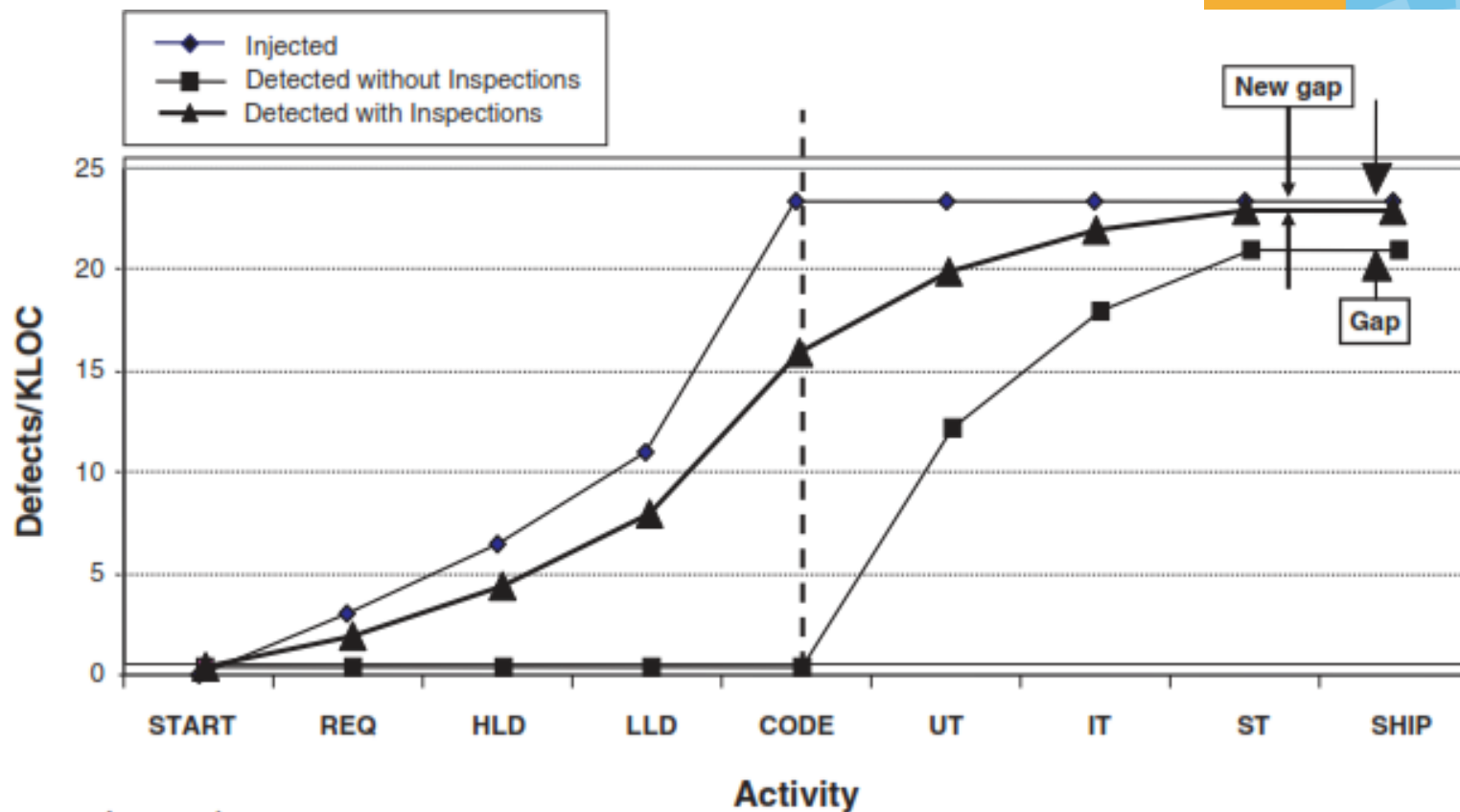


# Personal Review



Done by the person reviewing his own software product in order to find and fix the most defects possible.

- **Principles of a personal review**
  - Find and **correct all defects** in the software product.
  - Use a checklist produced from your **personal data**, if possible, using the **type of defects that you are already aware of**.
  - Follow a **structured** review process;
  - Use **measures** in your review;
  - Use **data to improve** your review;
  - Use **data to determine where and why defects were introduced** and then change your process to **prevent similar defects in the future**.



Legend:

- REQ = requirement
- HLD = high-level architecture design
- LLD = detailed design
- CODE = coding and debugging
- UT = unit testing
- IT = integration testing
- ST = system testing
- SHIP = delivery to customer
- KLOC = thousand lines of code

**Figure 5.5** Error detection during the software development life cycle [RAD 02].

## Checklist

A checklist is used as a memory aid. A checklist includes a list of criteria to verify the quality of a product. It also ensures consistency and completeness in the development of a task. An example of a checklist is a list that facilitates the classification of a defect in a software product (e.g., an oversight, a contradiction, an omission).

# Practices - to develop an effective and efficient personal review



- Pause between the development of a software product and its review.
- Examine products in hard copy rather than electronically.
- Check each item on the checklist.
- Update the checklists periodically to adjust to your personal data.
- Build and use a different checklist for each software product.
- Verify complex or critical elements with an in depth analysis.

### ENTRY CRITERIA

- None

### INPUT

- Software product to review

### ACTIVITIES

#### 1. Print:

- Checklist for the software product to be reviewed
- Standard (if applicable)
- Software product to review

2. Review the software product, using the first item on the checklist and cross this item off when the review of the software product is completed

3. Continue review of the software product using the next item on the checklist and repeat until all the items in the list have been checked

4. Correct any defects identified

5. Check that each correction did not create other defects.

### EXIT CRITERIA

- Corrected software product

### OUTPUT

- Corrected software product

### MEASURE

- Effort used to review and correct the software product measured in person-hours with an accuracy of +/- 15 minutes.

**Figure 5.6** Personal review process.

*Source:* Adapted from Pomeroy-Huff et al. (2009) [POM 09].

---

# THANK YOU