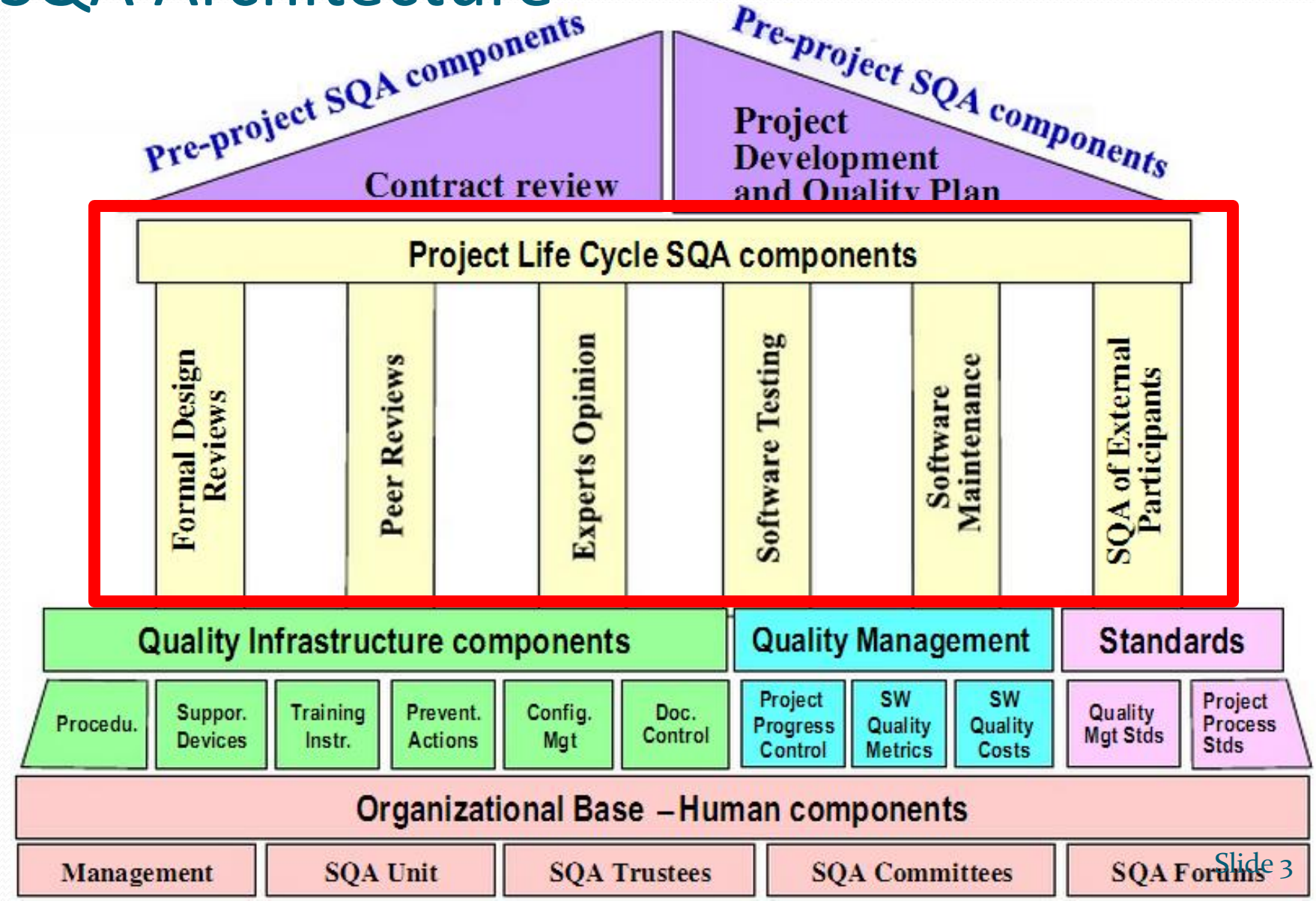
SQA components in the project life cycle

1 Overview	2 Life cycle components	3 Infrastructure components	4 Management components	5 Standards and Organizing
6 Static tesing	7 Dynamic testing	8 Test management	9 Tools	

Learning objectives

- Compare the three major **review** methodologies
- Explain the **fundamental test process**
- Distinguish **testing levels** and **testing types**
- List software **maintenance components** and explain their distinction
- Explain the benefits and risks with **external participants**
- List the SQA tools appropriate for use with external participants

SQA Architecture



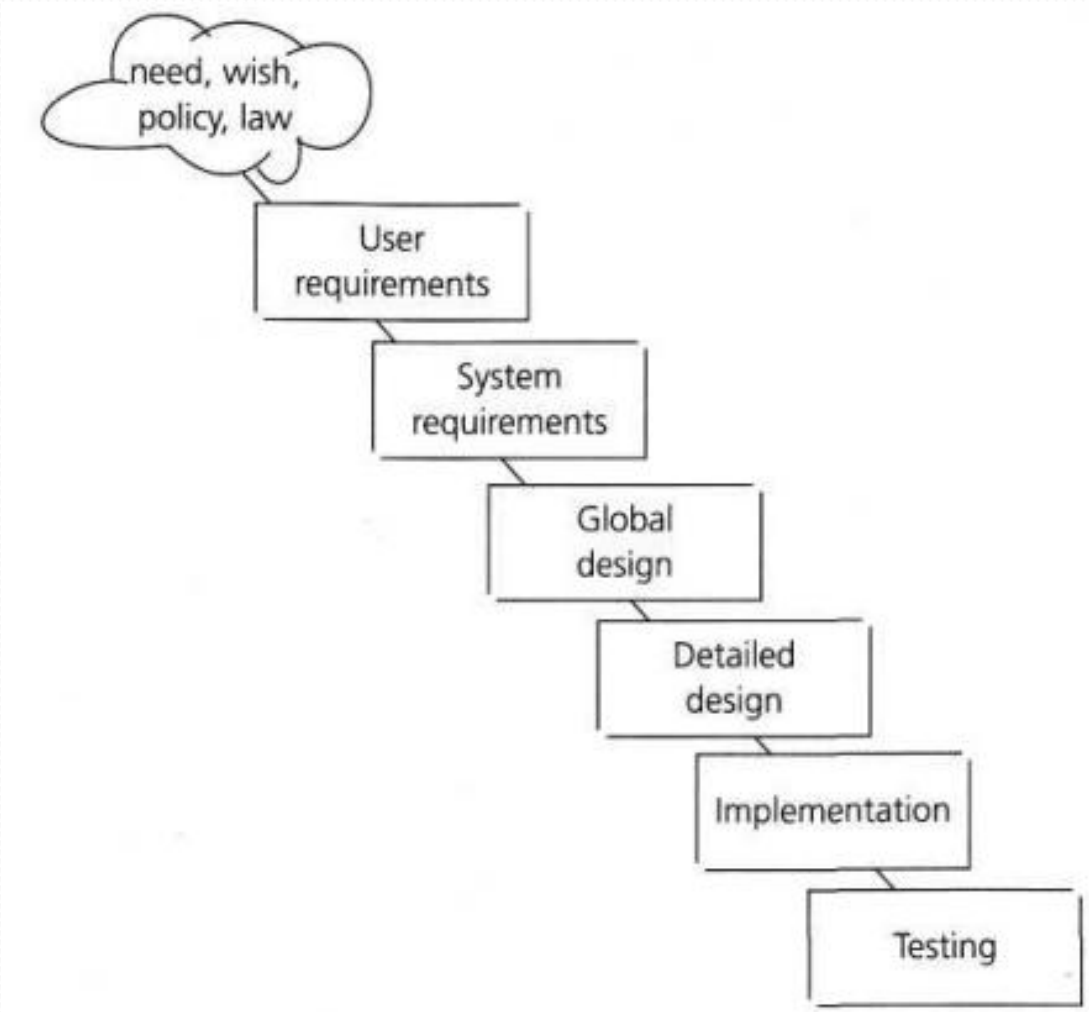
References

- Galin (2004). *Software Quality Assurance from theory to implementation.* Pearson Education Limited

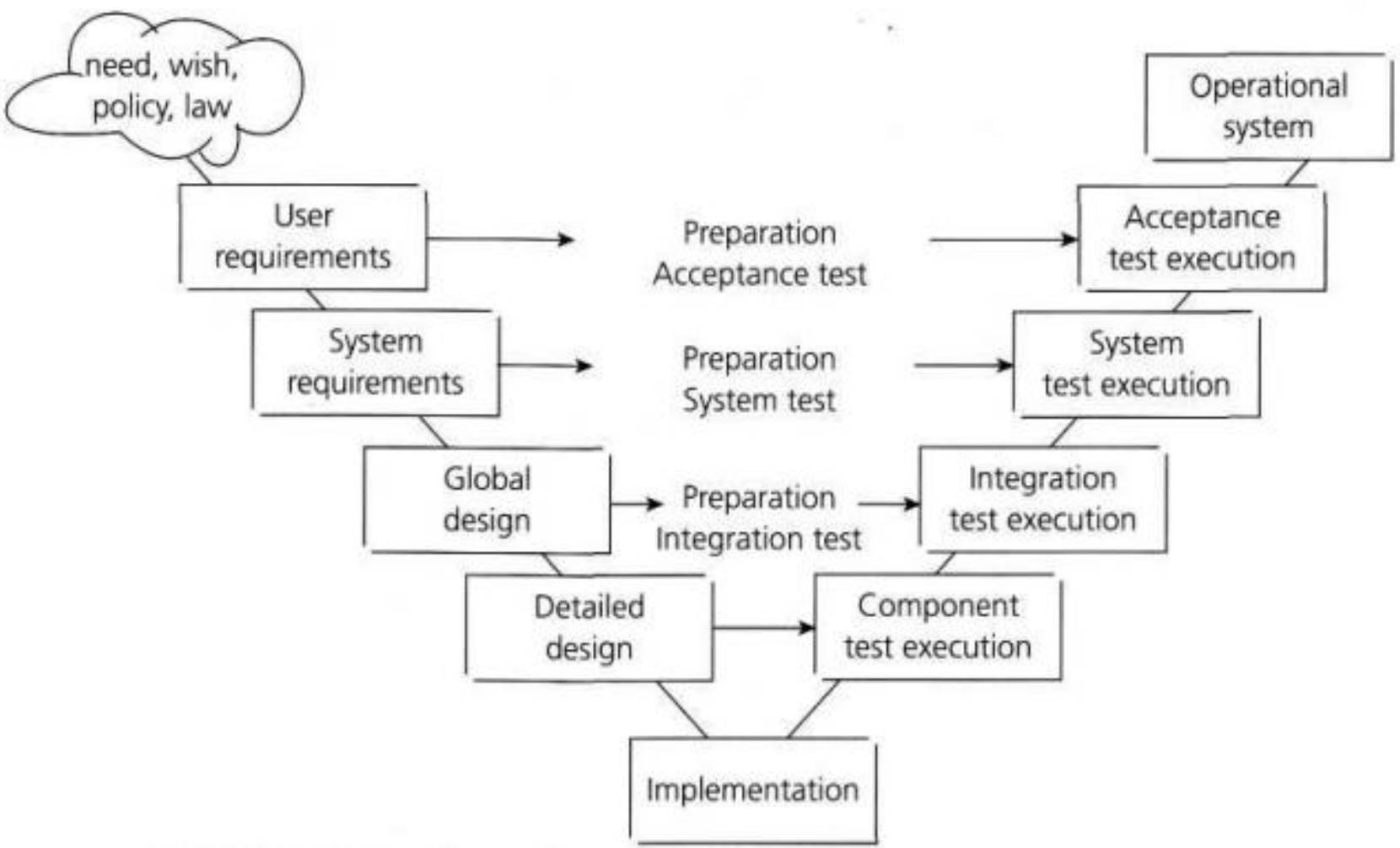
- Dorothy Grahamet, Erik van Veenendaal, Isabel Evans, Rex Black. *Foundations of software testing: ISTQB Certification*

Contents

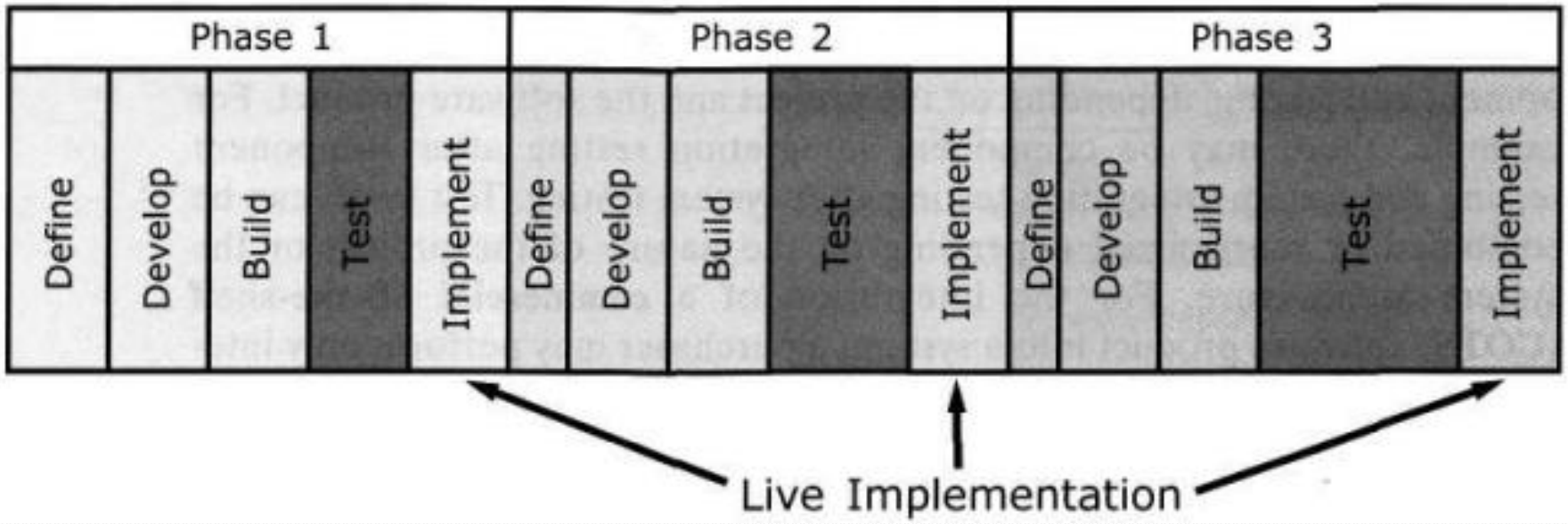
1. Quality activities in the project life cycle Waterfall model



1. Quality activities in the project life cycle V-model



1. Quality activities in the project life cycle Iterative development model



1. Quality activities in the project life cycle Testing within a life cycle model

- Several characteristics of good testing:
  - for every development activity there is a corresponding testing activity;
  - the analysis and design of tests for a given test level should begin during the corresponding development activity;
  - testers should be involved in reviewing documents as soon as drafts are available in the development cycle

Contents

1. Integrating quality activities in the project life cycle
2. **Review**
3. Software testing
4. Assuring the quality of software maintenance components
5. Assuring the quality of external participants' contributions

From ISTQB

2. Review

- The IEEE definition

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**

2. Review Objectives

- Direct
  - to **detect** analysis and design **errors**
  - to **identify new risks** likely to affect completion of the project
  - to **identify deviations** from templates and style procedures
  - to **approve** the analysis or design product
- Indirect
  - for exchange of professional knowledge
  - to record analysis and design errors that will serve as a basis for future *corrective actions*

2. Review
Types of reviews

- Walkthrough
- Inspection
- Technical review/Peer review

2. Review
Walkthrough

- An informal meeting for evaluation or informational purposes
- Led by the author
- The author describes his work product to a group of colleagues and solicits comments
- Goals
  - to present the document to stakeholders
  - to familiarize the audience with the content of the document
  - to gather feedback
  - to examine and discuss the validity of proposed solutions and the viability of alternatives, establishing consensus
- Especially useful if people from outside the software discipline or for higher-level documents

2. Review
Inspection

- The most formal review type
  - prescribed and systematic
  - roles are well defined (moderator, reader, and a recorder)
- Usually led by trained moderator (not the author)
- Attendees should prepare for this type of meeting by reading through the document
- The defects found are documented in a logging list
- The result of the inspection meeting should be a written report
- Metrics are gathered and analyzed to optimize the process

2. Review
Inspection (cont'd)

- Goals
  - to **find problems** and see what's missing, not to fix anything
  - **help** the author to improve the quality of the document, by producing documents with a higher level of quality
  - **train** new employees in the organization's development process
  - **learn** from defects found and **improve** processes in order to prevent recurrence of similar defects

2. Review
Technical review/Peer review

- A discussion meeting focuses on achieving **consensus about the technical content of a document**
- Includes **peer** and **technical experts** (e.g. architects, chief designers and key users), no management participation
- Goals
  - assess the **value of technical concepts** and **alternatives** in the product and project environment
  - establish **consistency in the use and representation** of technical concepts
  - ensure that **technical concepts are used correctly**

Contents

3. Software testing

3.1. Objectives

3.2. Fundamental test process

3.3. Test levels

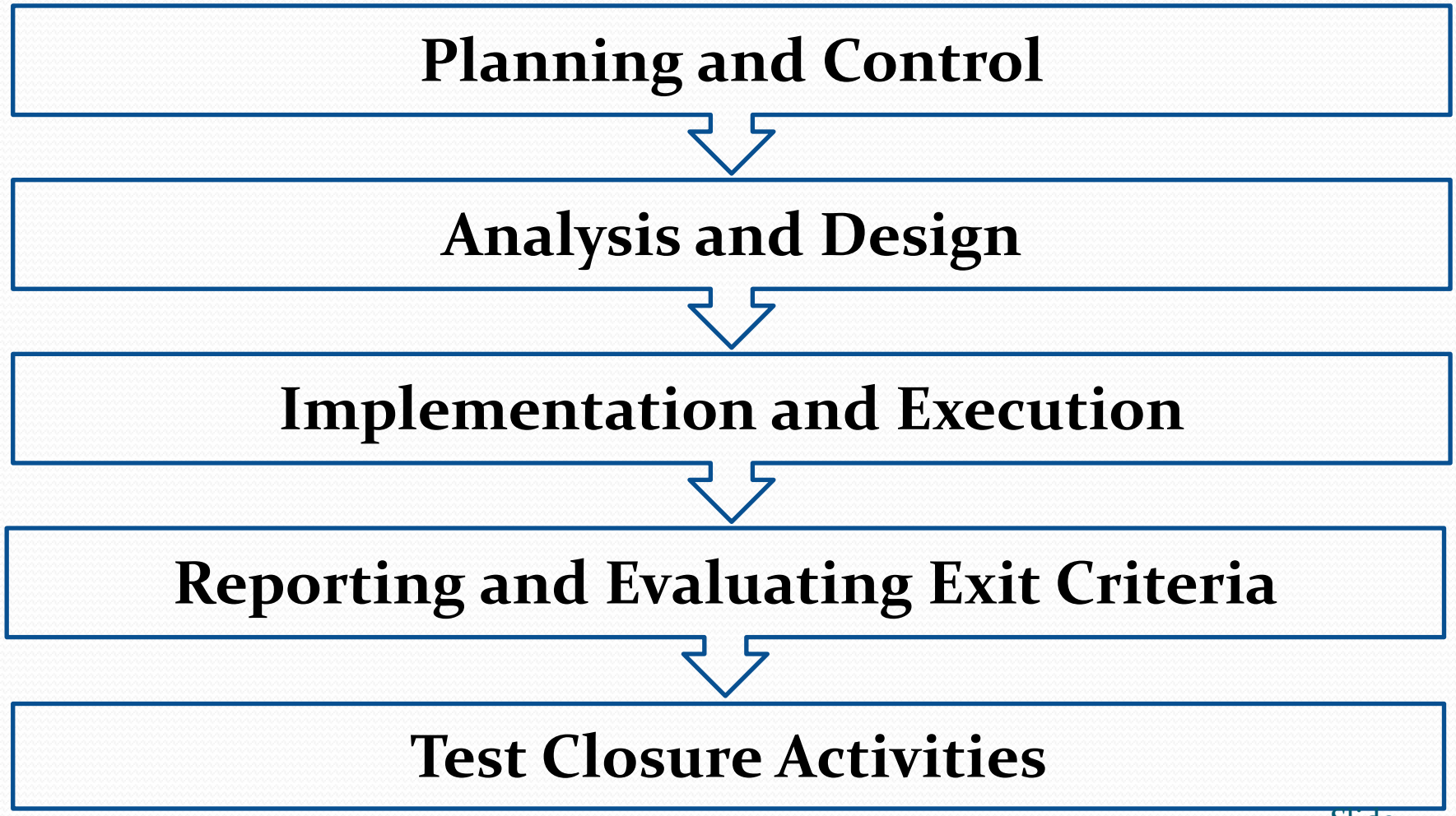
3.4. Test types

From ISTQB

3.1. Testing objectives

- to detect defects
- to determine that they satisfy specified requirements
- to demonstrate that they are fit for purpose

3.2. Test process



3.2. Test process
Test planning and control

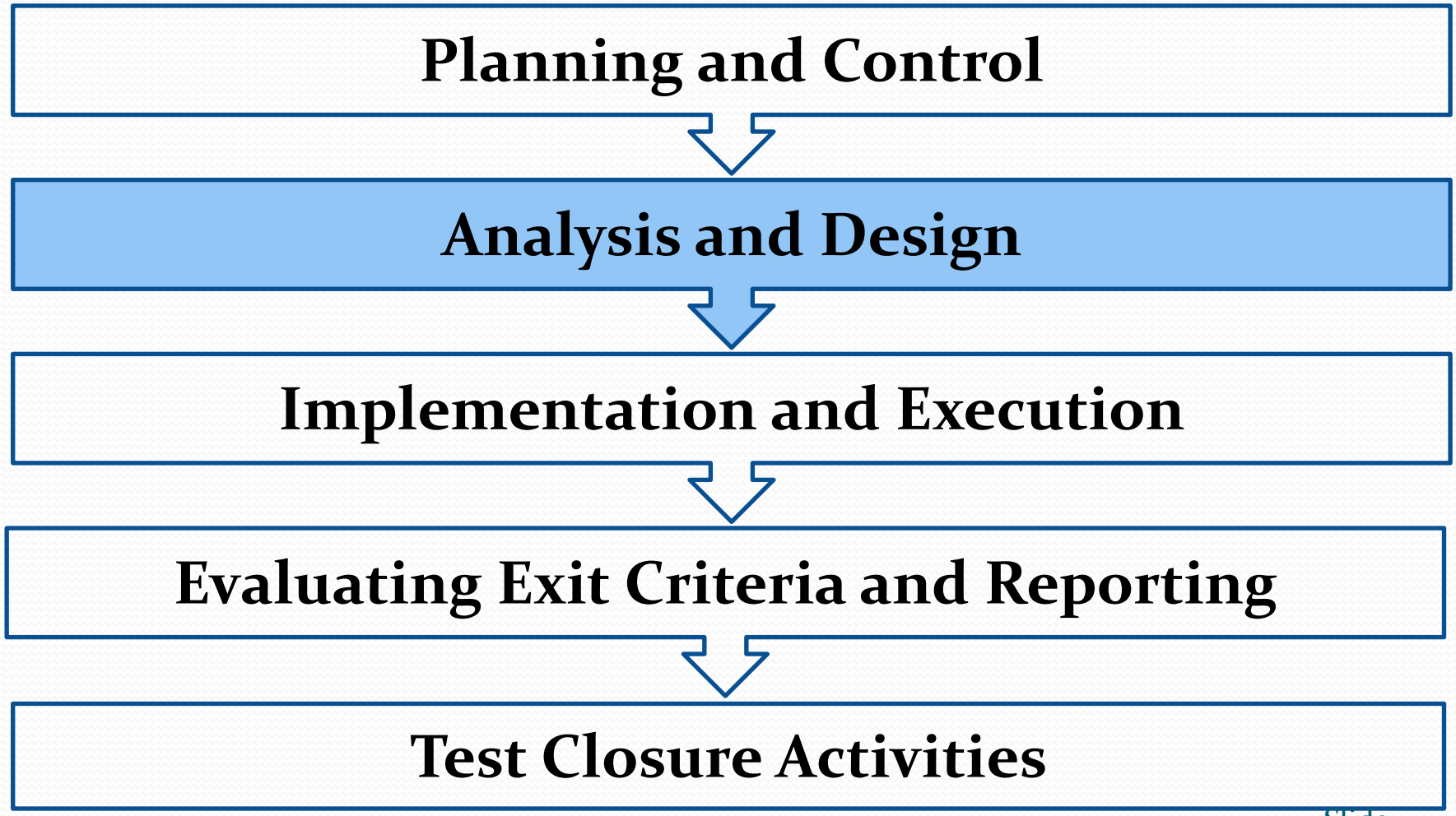
- Test planning has the following major tasks:
  - *determine* the **scope**, **risks** and the **objectives** of testing
  - *determine* the **test approach** (techniques test items, coverage, identifying and interfacing with the teams involved in testing, testware)
  - *determine* the required test **resources** (e.g. people, test environment, PCs)
  - ***schedule*** test analysis and design tasks, test implementation, execution and evaluation
  - *determine* the **exit criteria** to know when testing is finished

Template_Test Plan

3.2. Test process
Test planning and control (cont'd)

- Test control is an ongoing activity
- Monitor status and take corrective action if necessary
- Test control has the following major tasks:
  - *measure* and *analyze* the results of reviews and testing
  - *monitor* and *document* progress, test coverage and exit criteria
  - *provide information* on testing
  - *initiate* **corrective actions**
  - *make decisions*: to continue testing, to stop testing, to release the software or to retain it for further work

3.2. Test process



3.2. Test process
Test analysis and design

- Test analysis and design is the activity where general testing objectives are transformed into tangible **test conditions** and **test cases**

3.2. Test process
Test analysis and design

- Major tasks:

  - *review* the **test basis** (such as requirements, architecture, design, interfaces), *examining* the **specifications** for the software under test

  we can start designing certain kinds of tests before the code exists (called black-box tests), as we can use the test basis documents to understand what the system should do once built

3.2. Test process
Test analysis and design

- Major tasks:
  - *evaluate* **testability** of the test basis and test objects, e.g.

    - if the requirements just say 'the software needs to respond quickly enough' that is not testable, because 'quick enough' may mean different things to different people

    - a more testable requirement would be 'the software needs to respond in 5 seconds with 20 people logged on'

3.2. Test process
Test analysis and design

- Major tasks:

  - *Identifying* and *prioritizing* **test conditions**, based on analysis of test items, the specifications, behavior and structure, i.e. determine ‘what’ is to be tested

    this gives us a high-level list of what we are interested in testing, e.g.

    “number items ordered > 99”

    “the ID must be numeric”

3.2. Test process
Test analysis and design

- Major tasks:
  - *Designing* and *prioritizing* **test cases** (using techniques to help select representative tests which carry risks or which are of particular interest) based on the test conditions and going into more detail

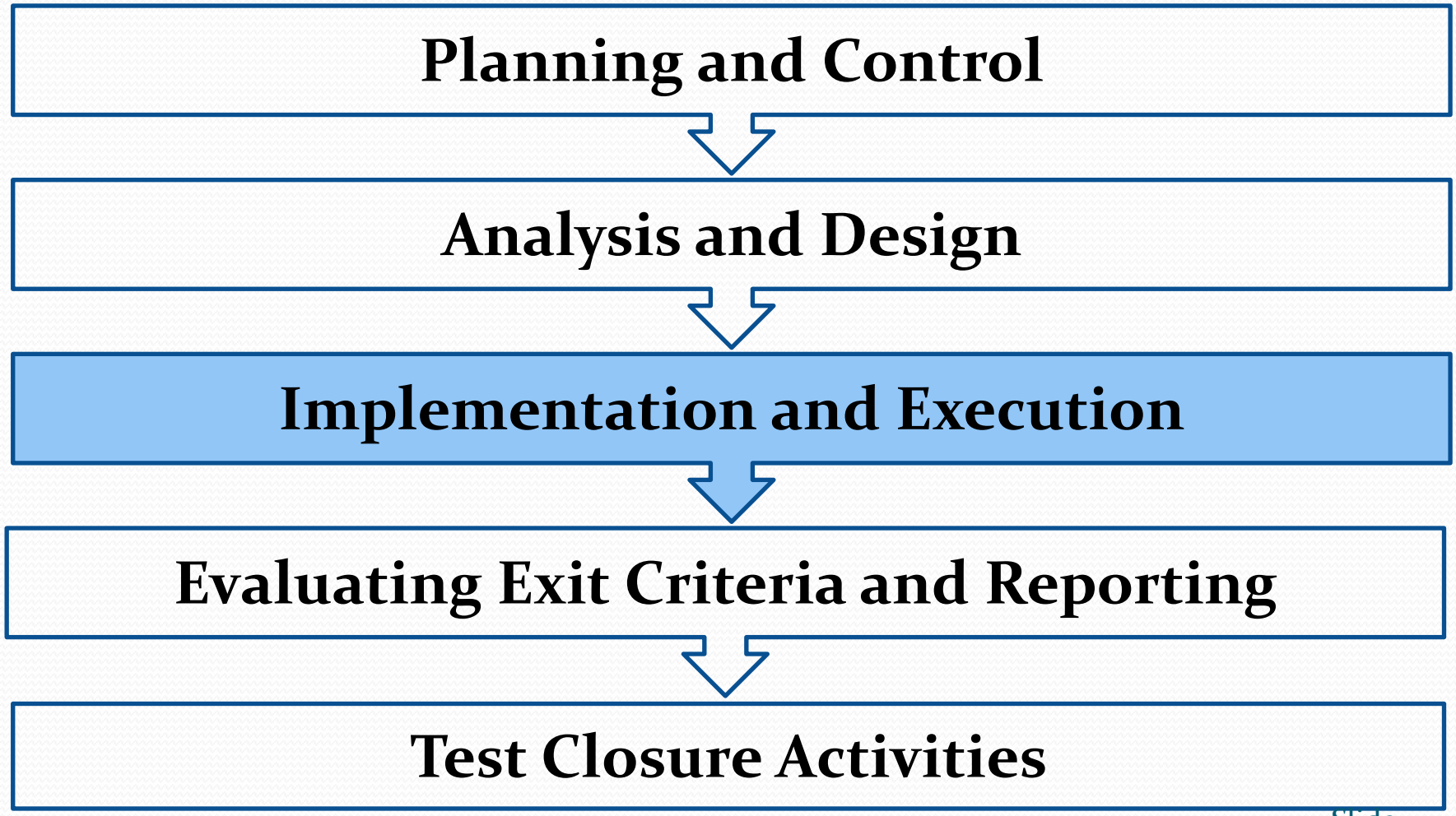
3.2. Test process
Test analysis and design

- Major tasks:

  - *design* the **test environment** set-up and *identify* any required **infrastructure** and **tools**

    this includes testing tools and support tools such as spreadsheets, word processors, project planning tools, and non-IT tools and equipment - everything we need to carry out our work

3.2. Test process



3.2. Test process
Test implementation and execution

- Major tasks:
  - *develop* and *prioritize* test cases, create **test data** for those tests, write **test procedures**
  - *create* **test suites** from the test cases for efficient test execution (a test suite is a logical collection of test cases which naturally work together)
  - *set up* a **test execution schedule**
  - *verifying* that **the test environment has been set up correctly**, possibly even running specific tests on it

3.2. Test process
Test implementation and execution

- Major tasks:
  - *execute* the test suites and individual test cases, following the test procedures
    - do this manually or by using test execution tools, according to the planned sequence
  - *log* the outcome of test execution, *record* the identities and ***versions of the software under test***, test tools and **testware**
  - *compare* actual results with expected results

3.2. Test process
Test implementation and execution

- Major tasks:
  - where there are differences between actual and expected results, *reporting discrepancies* as **incidents** with as much information as possible, including possible causal analysis (e.g. a defect in the code, in specified test data, in the test document, or a mistake in the way the test was executed)

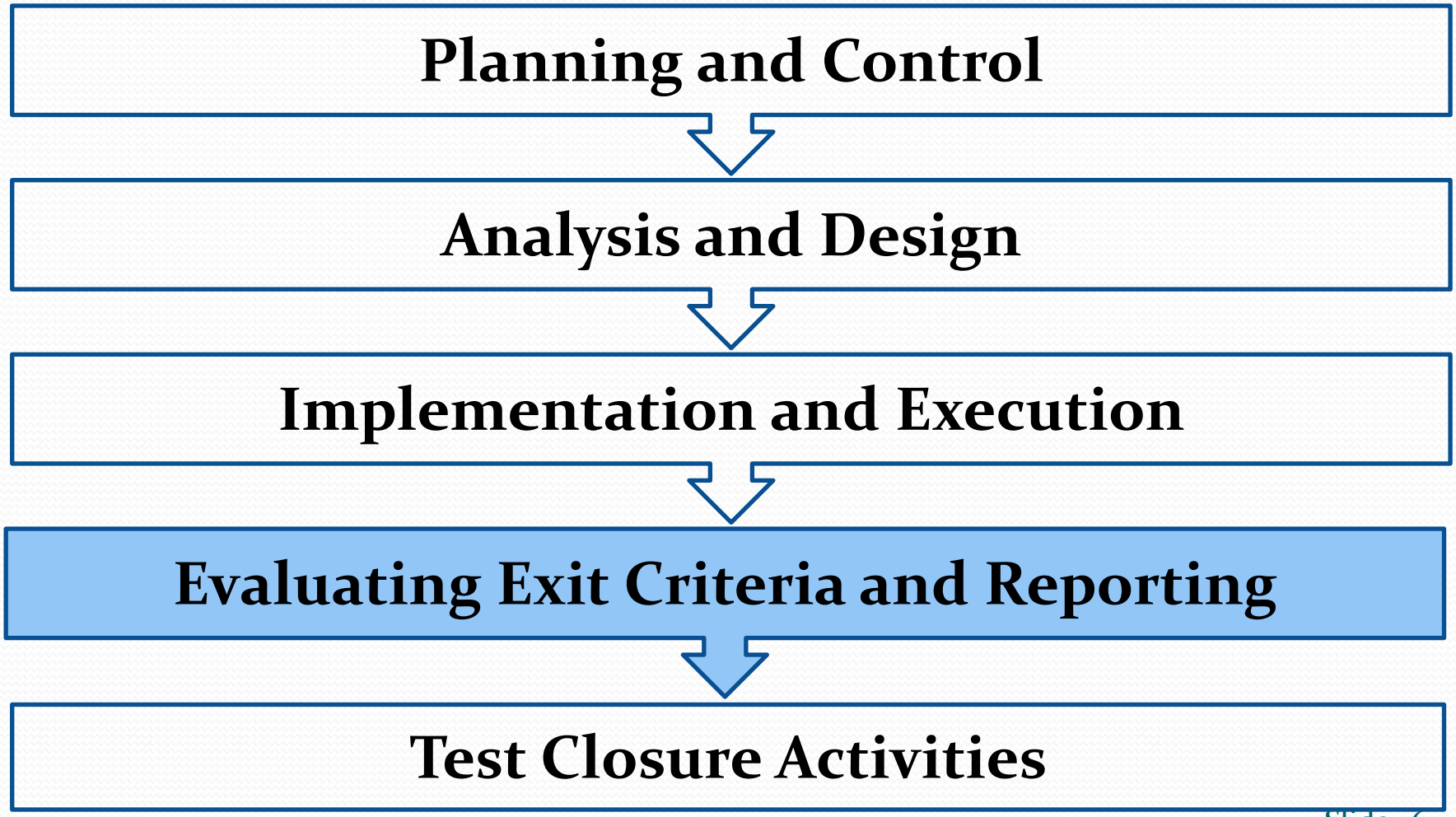
3.2. Test process
Test implementation and execution

- Major tasks:
  - *repeat test activities* when changes have been made following incidents

    re-execute tests that previously failed in order to confirm a fix (**confirmation testing** or **re-testing**)

    execution of a corrected test and/or execution of tests in order to ensure that defects have not been introduced in unchanged areas of the software or that defect fixing did not uncover other defects (**regression testing**)

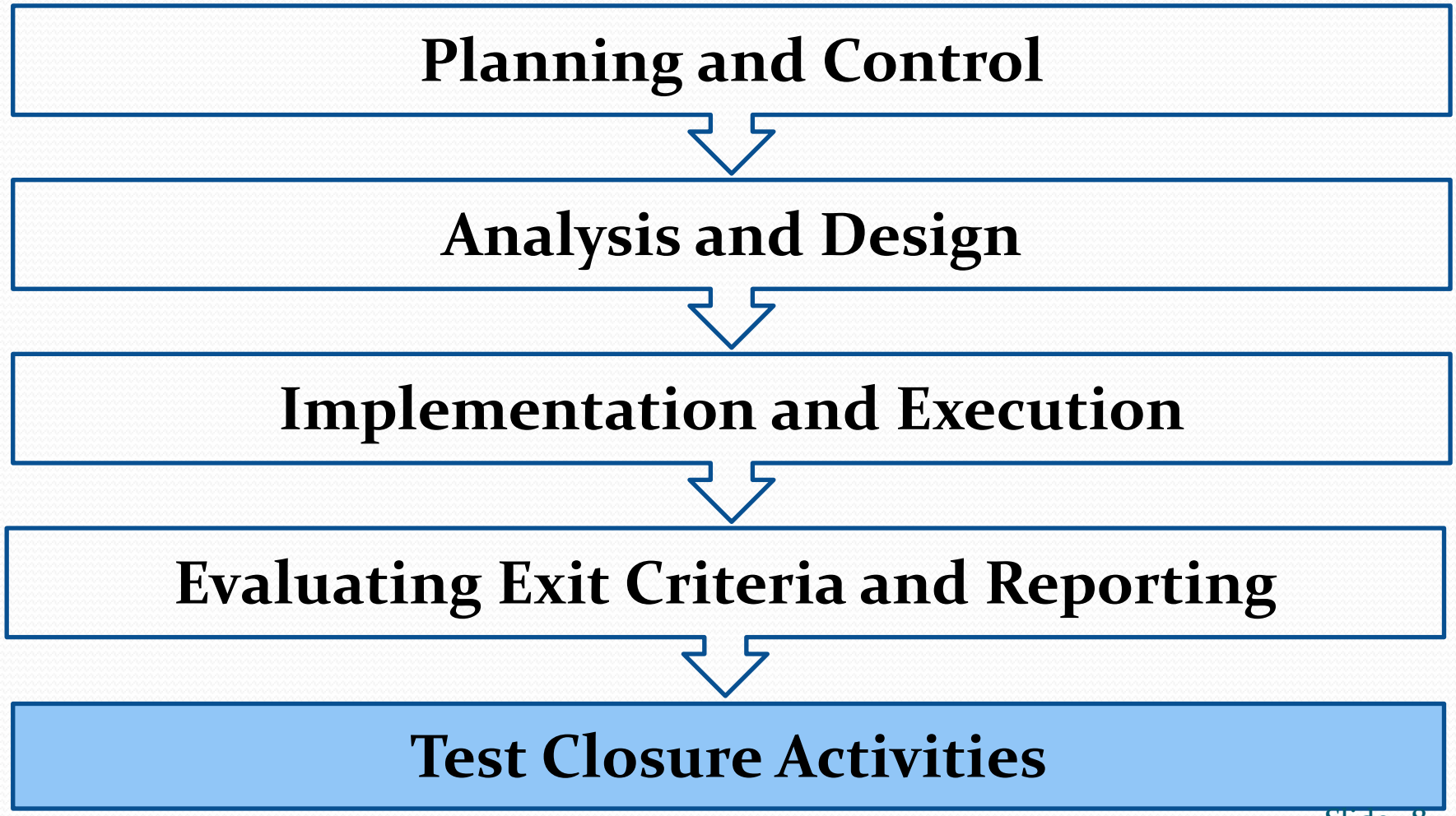
3.2. Test process



3.2. Test process
Evaluating exit criteria and reporting

- Evaluating exit criteria is the activity where **test execution is assessed against the defined objectives,** to know whether we have done enough testing
- Major tasks:
  - *check* test logs against the exit criteria specified in test planning
  - *assess* if more tests are needed or if the exit criteria specified should be changed
  - *write* a **test summary report** for stakeholders

3.2. Test process



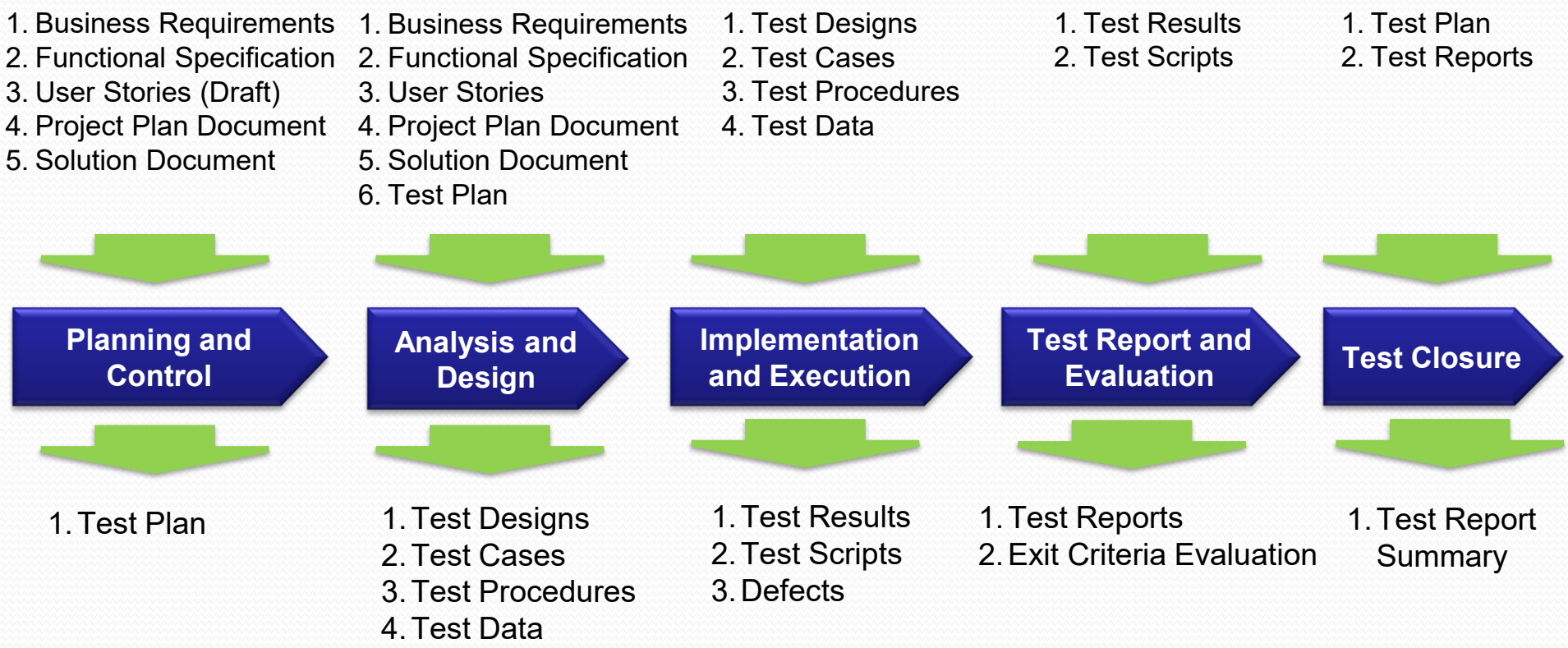
3.2. Test process
Test closure activities

- Collect data from completed test activities to consolidate experience
- We may need to do this when:
  - software is delivered
  - have gathered the information needed from testing
  - the project is cancelled
  - a particular milestone is achieved

3.2. Test process
Test closure activities

- Major tasks:
  - *check* which planned deliverables we actually delivered and ensure all incident reports have been resolved
  - *finalize* and *archive* **testware** (such as scripts, test environment, and any other test infrastructure) for later reuse
  - *hand over* testware to the maintenance organization
  - *analyzing lessons learned* for future releases and projects, and incorporating lessons to improve the testing process

3.2. Test process Inputs & Outputs



Kiểm tra hiểu bài
Quy trình kiểm thử

1. Việc đánh giá **tính kiểm thử được** của yêu cầu và hệ thống thuộc giai đoạn nào.
2. Chuẩn bị môi trường và các công cụ kiểm thử cần thiết là các công việc thuộc giai đoạn nào.
3. Giai đoạn nào phải xác định xem có cần thêm test.
4. Báo cáo sự cố kiểm thử thuộc giai đoạn nào.
5. Cần có những tài liệu gì để viết test plan.
6. Sau khi thực thi kiểm thử, các tester thường làm gì.

Kiểm tra hiểu bài
Quy trình kiểm thử

7. Which of the following requirements is testable?

   A. The system shall be user friendly

   B. The safety-critical parts of the system contain 0 faults

   C. The response time shall be less than one second for the specified design load

   D. The system shall be built to be portable

8. Which option is part of the 'implementation and execution' area of the fundamental test process?

   A. Developing the tests

   B. Comparing actual and expected results

   C. Writing a test summary

   D. Analysing lessons learnt for future releases

3.3. Test levels

- There are 4 levels of testing:
  - Component test (Unit test)
  - Integration test
  - System test
  - Acceptance test

3.3. Test levels
Component test [1/3]

- Individual software components (e.g. program, function, procedure, method, object, etc.) are tested separately
- Test basis:
  - component requirements
  - detailed design
  - code
- Done by: development team
- When?
  - After coding
  - Before integration test

3.3. Test levels
Component test [2/3]

- Component test may include:
  - Testing of **functionality**
  - Testing of specific **non-functional** characteristics, such as resource-behaviour (e.g. memory leaks) or performance testing
  - **Structural** testing (structure-based techniques)

3.3. Test levels
Integration test

- Test **interfaces** between components, **interactions** to different parts of a system such as an operating system, file system and hardware or interfaces between systems
  - Component integration test:

    Tests the interactions between software components and is done after component testing
  - System integration test:

    Tests the interactions between different systems and may be done after system testing
- Done by: carried out by the developers, but preferably by a specialist integration testers

3.3. Test levels
Integration test (cont.)

- Test basis:
  - software and system design
  - architecture
  - workflows
  - use cases
- Integration test may include:
  - Testing of functionality
  - Testing of specific non-functional characteristics (e.g. performance)
  - Structural testing (Structure-based techniques)
- Integration strategy: **Big-bang** vs **Incremental** (Top-down, Bottom-up, Functional)

3.3. Test levels
Integration test – Big-bang

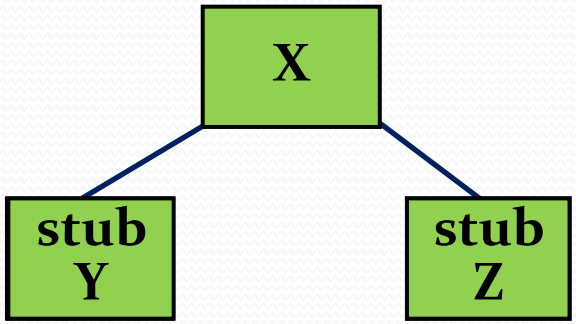
- Strategy
  - all tested components or systems are **integrated simultaneously**, after which everything is tested as a whole
- Advantage
  - everything is finished before integration testing starts
  - no need to simulate
- Disadvantage
  - it is **time-consuming** and **difficult to trace the cause of failures**
    - takes longer to **locate** and **fix faults**
    - **re-testing** after fixes **more extensive**

3.3. Test levels
Integration test – Incremental

- Strategy
  - all components are **integrated one by one**, and a test is carried out **after each step**
    - Baseline 0: tested component
    - Baseline 1: two components
    - Baseline 2: three components, etc.
- Advantages
  - **defects are found early** in a smaller assembly
  - relatively easy to detect the cause
- Disadvantage
  - time-consuming to develop **stubs** and **drivers**

3.3. Test levels
Integration test – Incremental (cont.)

- Stub replaces a **called component** for integration testing
- For example: if we have modules X, Y, Z (X calls functions from Y and Z)



X is **ready to be tested**

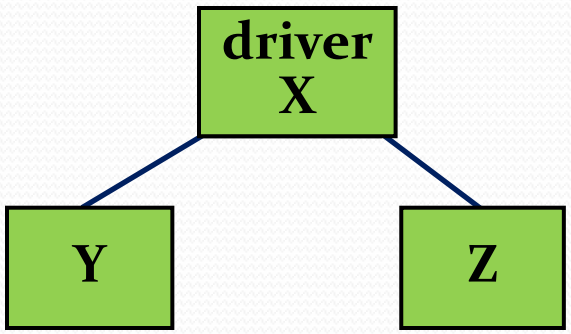
Y and Z are **not ready**

write stubs to simulate Y and Z to return values for X

```
public int generateRandInt()
{
    return 1; // eventually need to return a random integer 1..100
}
```

3.3. Test levels
Integration test – Incremental (cont.)

- Driver which **calls a component** to be tested
- For example: if we have modules X, Y, Z, and we need to test Y and Z modules



X is **not ready**

Y and Z are **ready to be tested**

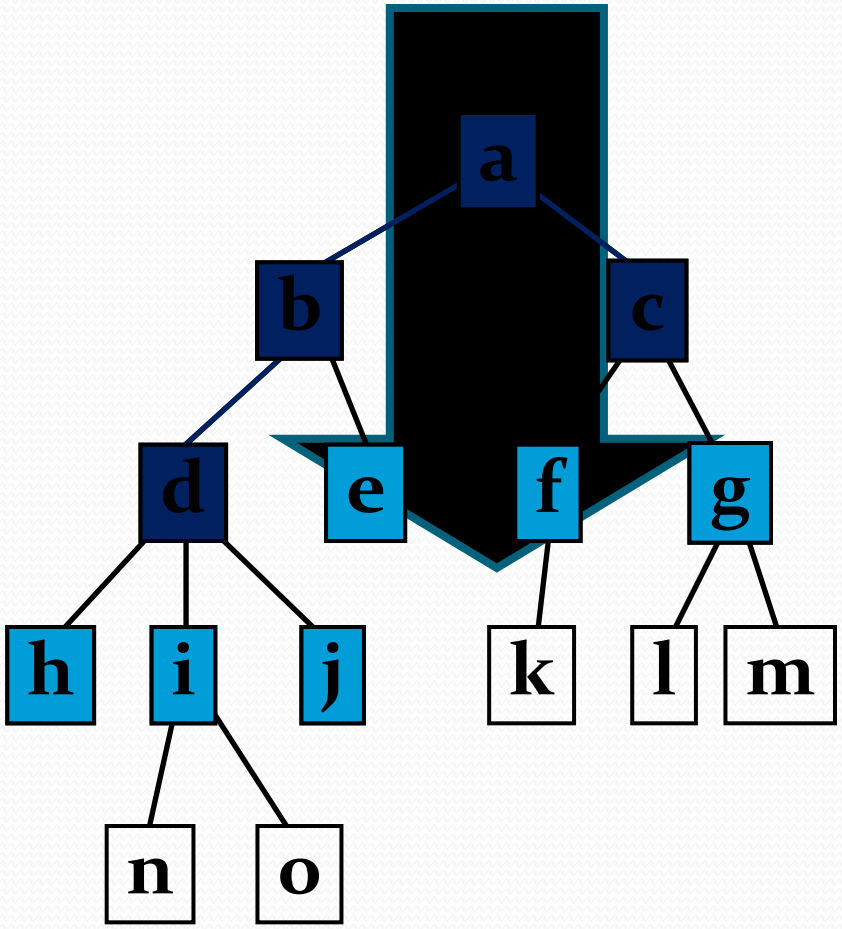
write driver for X, which returns values for Y and Z

3.3. Test levels
Integration test – Incremental (cont.)

- Methods:
  - **top-down**: testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu)
  - **bottom-up**: testing takes place from the bottom of the control flow upwards
  - **functional incremental**: integration and testing **takes place on the basis of the functions**, as documented in the functional specification

3.3. Test levels
Integration test – Top-down

- Baselines:
  - baseline 0: component a
  - baseline 1: a + b
  - baseline 2: a + b + c
  - baseline 3: a + b + c + d
  - etc.
- Need to call to lower level components not yet integrated
- Need stubs to replace a called component



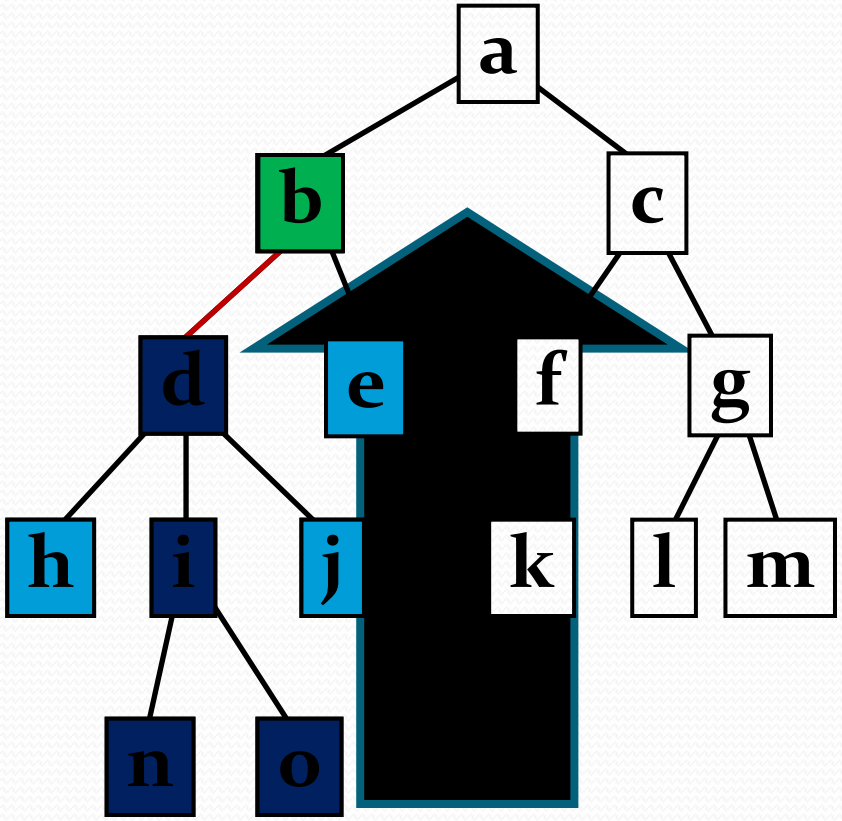
3.3. Test levels
Integration test – Top-down (cont.)

- Advantages:
  - critical control structure tested first and most often
  - can demonstrate system early (show working menus)

- Disadvantages:
  - needs stubs
  - detail left until last

3.3. Test levels
Integration test – Bottom-up

- Baselines:
  - baseline 0: component n
  - baseline 1: n + i
  - baseline 2: n + i + o
  - baseline 3: n + i + o + d
  - etc.
- Also needs stubs for some baselines
- Needs drivers to call the baseline configuration



3.3. Test levels
Integration test – Bottom-up (cont.)

- Advantages:
  - lowest levels tested first and most thoroughly
  - visibility of detail

- Disadvantages
  - no working system until last baseline
  - needs both drivers and stubs
  - major control problems found last

3.3. Test levels
System test

- The process of testing an integrated system to **verify** that it meets specified requirements
- Test basis:
  - requirements specification
  - business processes
  - use cases
  - risk analysis reports
- **The test environment should correspond to the final target environment as much as possible**
- Done by: a third party team or by business analysts

3.3. Test levels
System test (cont.)

- Should investigate both **functional** and **non-functional requirements** of the system
  - functional requirements
    - using **specification-based techniques** (e.g. create decision table in business rules,…)
    - using **structure-based techniques** (e.g. the coverage of menu options, web page navigation,…)
  - non-functional requirements
    - e.g. performance, reliability, usability, configuration/installation, back-up/recovery...

3.3. Test levels

Acceptance test

- Most often focused on a **validation** type of testing
- Most often the responsibility of the **user** or **customer**
- Finding defects is not the main focus in acceptance testing
- Test basis:
  - user requirements
  - system requirements
  - use cases
  - business processes
  - risk analysis reports
- Done by: Customer/independent test team

3.3. Test levels
Acceptance test (cont.)

- Typical forms of acceptance testing:
  - user acceptance test
  - operational acceptance test
  - contract and regulation acceptance test
  - alpha test/beta test

3.3. Test levels

Acceptance test – User acceptance test

- Typically verifies the **fitness for use** of the system by business users
  - customer (user) should perform or be closely involved
  - customer can perform any test they wish, usually based on their business processes
- Why customer/user involvement
  - Users know:
    - what really happens in business situations
    - complexity of business relationships
    - how users would do their work using the system
    - examples of real cases
    - how to identify sensible work-arounds

3.3. Test levels

Acceptance test – Operational acceptance test

- Also called production acceptance test
- Validates whether the system meets the requirements for operation, often performed by the **system administrators**
- May be included:
  - testing of backup/restore
  - disaster recovery
  - user management
  - maintenance tasks
  - data load and migration tasks
  - periodic checks of security vulnerabilities

3.3. Test levels

Acceptance test – Contract/regulation acceptance test

- Contract acceptance testing is **performed against a contract's acceptance criteria** and any documented agreed changes

- Regulation acceptance testing (compliance acceptance testing) is **performed against the regulations which must be adhered to**, such as governmental, legal or safety regulations

3.3. Test levels
Acceptance test – Alpha and Beta test

- Similarities:
  - Use for the systems that has been developed for the mass market, e.g. commercial off-the-shelf software (COTS)
  - When software is stable
  - Use the product in a realistic way in its operational environment
  - Give comments back on the product
    - faults found
    - how the product meets their expectations
    - improvement/enhancement suggestions

3.3. Test levels
Acceptance test – Alpha and Beta test

- Differences:
  - Alpha testing
    - simulated or actual operational testing performed **at the developer's site**
    - potential users and members of the developer's organization are invited to use the system
  - Beta testing (field testing)
    - performed by customers or potential customers **at their own locations**

4. Test types

- A test type is focused on a particular test objective:
  - functional test
  - non-functional test
  - structural test
  - testing related to changes (i.e. confirmation test, regression test)

4. Test types
Functional test

- Testing **functions** of a system or component
  - '**what**' the system does
- The functions typically **described in work products** (*requirements specification, functional specification, use cases*) or they may be undocumented
- May be performed at all test levels
- Testing functionality can be done from two perspectives:
  - requirements-based
  - business process-based

4. Test types
Functional test – Requirements-based

- The basis for designing tests: **specification of the functional requirements** for the system
  - use **table of contents of the requirements specification** as an initial list of items to test (or not to test)
  - **prioritize the tests based on prioritized requirements in the specification** (also prioritize the requirements based on risk criteria if not already done) to ensure that the most important and most critical tests are included in the testing effort

4. Test types
Functional test – Business process-based

- Uses knowledge of the business processes
  - Business scenarios
    - involved in the day-to-day business use of the system
  - Use cases
    - prepared cases based on real situations

4. Test types
Non-functional test

- The testing of a **non-functional quality characteristic**
  - **'how well'** the system works, measure on a scale of measurement, e.g. time to respond
- May be performed at all test levels
- Non-functional testing includes: performance testing, load testing, stress testing, usability testing, maintainability testing, reliability testing, portability testing…

4. Test types
Structural test

- Referred to as 'white-box' or 'glass-box‘
- May be performed at all test levels
  - it tends to be mostly applied at **component** and **integration**
  - can also be applied at **system** or **acceptance testing** levels (e.g. business models or menu structures)

4. Test types
Testing related to changes

- If you have made a change to the software, you will have changed the way it functions, the way it performs (or both) and its structure
- The specific types of tests relating to changes may be **included all of the other test types**
  - confirmation testing (re-testing)
  - regression testing

4. Test types – Testing related to changes Confirmation testing (Re-testing)

- Re-execute test after faults are fixed
  - run a test, it fails, the defect is reported
  - new version of software with defect fixed
  - re-run the test to **confirm that the defect has indeed been fixed**
- Re-run the same test
  - must be exactly repeatable
  - same environment, inputs and preconditions
- If the test now passes, this mean that
  - **the software is correct?**
  - **introduced a different defect elsewhere**?

4. Test types – Testing related to changes Regression testing

- Executing test cases that have been executed before
  - include the tests that probably passed the last time executed
- Regression testing are performed
  - whenever **the software changes**, either as a result of fixes or new or changed functionality
  - when **the environment changes**, even if application functionality stays the same, e.g. new version of DBMS
  - for **emergency fixes** (possibly a subset)

*Basic concepts
Test basis, test condition

- Test basis
  - all documents from which the requirements of a component or system can be inferred
  - e.g. requirements, architecture, design, interfaces

- Test condition
  - an item or event of a component or system that could be verified by one or more test cases
  - e.g. function, transaction, quality attribute

*Basic concepts
Test case, test suite

- Test case:
  - a set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition

- Test suite:
  - a set of several test cases for a component or system under test, where the post condition of one test is often used as the precondition for the next one

*Basic concepts
Verification, validation

- Verification:
  - verify whether the product **meets the requirements set**
  - focuses on the question 'Is the deliverable built according to the specification?'
- Validation:
  - checks whether the product **meets the user needs and requirements**
  - focuses on the question 'Is the deliverable fit for purpose, e.g. does it provide a solution to the problem?'

*Basic concepts
Static testing, dynamic testing

- Static testing: testing without running the code
  - static testing is performed using the software documentation (e.g. specification)
  - most verification techniques are static tests (e.g. feasibility reviews, requirements reviews)

- Dynamic testing: requires the code to be executed to perform the tests
  - most validation tests are dynamic tests (e.g. unit testing, integrated testing,...)

Contents

1. Integrating quality activities in the project life cycle
2. Reviews
3. Software testing
4. **Assuring the quality of software maintenance components**
5. Assuring the quality of external participants' contributions

4. Software maintenance components

- Corrective maintenance
  - for software corrections and user support services
- Adaptive maintenance
  - adjusts the software package to the requirements of new customers and changing environmental conditions
- Functionality improvement maintenance
  - combines perfective maintenance and preventive maintenance

4. Software maintenance components The foundation of high quality

- Foundation 1: Software package quality
- Foundation 2: Maintenance policy

4. Software maintenance components
Foundation 1: Software package quality

- Making code readable and maintainable is as important, or more important than making it work correctly
  - if it doesn't work, it can be fixed. If it can't be maintained, it's scrap
- From McCall quality factor, factors that have direct impact on software maintenance:
  - product operation factors: Correctness, Reliability
  - product revision factors: Maintainability, Flexibility, Testability
  - product transition factors: Portability, Interoperability

4. Software maintenance components Foundation 1: Software package quality

- Correctness
  - output correctness
    - accuracy, completeness of required output
    - up-to-dateness, availability of the information
  - documentation correctness
    - the quality of documentation: completeness, accuracy, documentation style and structure
  - coding qualification
    - compliance with coding instructions, especially those that limit and reduce code complexity and define standard coding style
- Reliability
  - the frequency of system failures as well as recovery times

4. Software maintenance components Foundation 1: Software package quality

- Maintainability
  - by following the software structure and style requirements
  - by implementing programmer documentation requirement
- Flexibility
  - there's more space for future functional improvement
- Testability
  - includes the availability of system diagnostics to be applied by the user as well as failure diagnostics to be applied by the support center or the maintenance staff at the user's site

4. Software maintenance components Foundation 1: Software package quality

- Portability
  - the software's potential application in different hardware and operating system environments, including the activities that enable those applications
- Interoperability
  - capacity to operate with other packages and computerized equipment

4. Software maintenance components
Foundation 1: Software package quality

Quality factor	Quality sub-factors	Software maintenance components		
		Corrective	Adaptive	Functionality improvement
Correctness	Output correctness	High		
	Documentation correctness	High	High	High
	Coding qualification	High	High	High
Reliability		High		
Maintainability		High	High	High
Flexibility			High	
Testability		High		
Portability			High	
Interoperability			High	

4. Software maintenance components Foundation 2: Maintenance policy

- Version development policy
  - sequential version policy
    - only one version is made available to the entire customer population, serve the needs of all customers
    - the new version replaces the current version
  - tree version policy
    - develop a specialized, targeted version for groups of customers or a major customer
- Change policy
  - the method of examining each change request and the criteria used for its approval

4. Software maintenance components Pre-maintenance quality components

- Maintenance contract review
  - activities:
    - proposal draft reviews
    - contract draft reviews
  - objectives
    - customer requirements clarification
    - review of alternative approaches to maintenance provision
    - review of estimates of required maintenance resources
    - review of maintenance services to be provided by subcontractors and/or the customer
    - review of maintenance costs estimates

4. Software maintenance components Pre-maintenance quality components

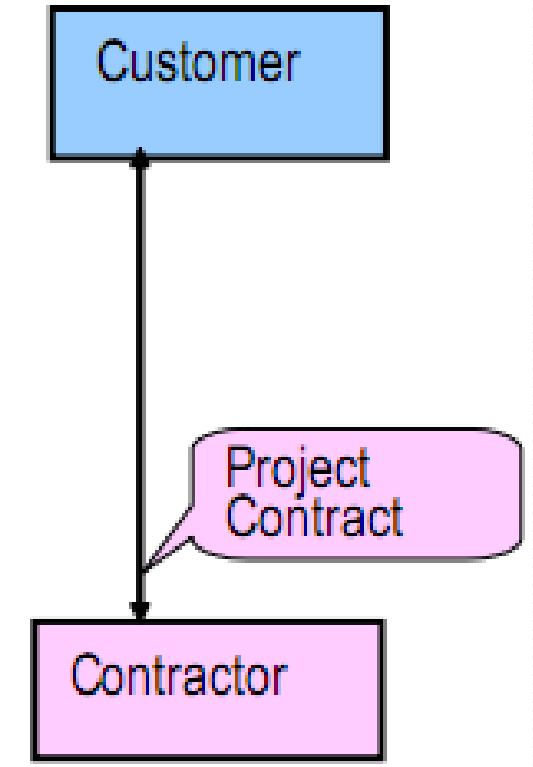
- Maintenance plan construction
  - should be prepared for all customers, external and internal
  - the plan includes
    - a list of the contracted maintenance services
    - a description of the maintenance team's organization
    - a list of maintenance facilities
    - a list of identified maintenance service risks
    - a list of required software maintenance procedures and controls
    - the software maintenance budget

Contents

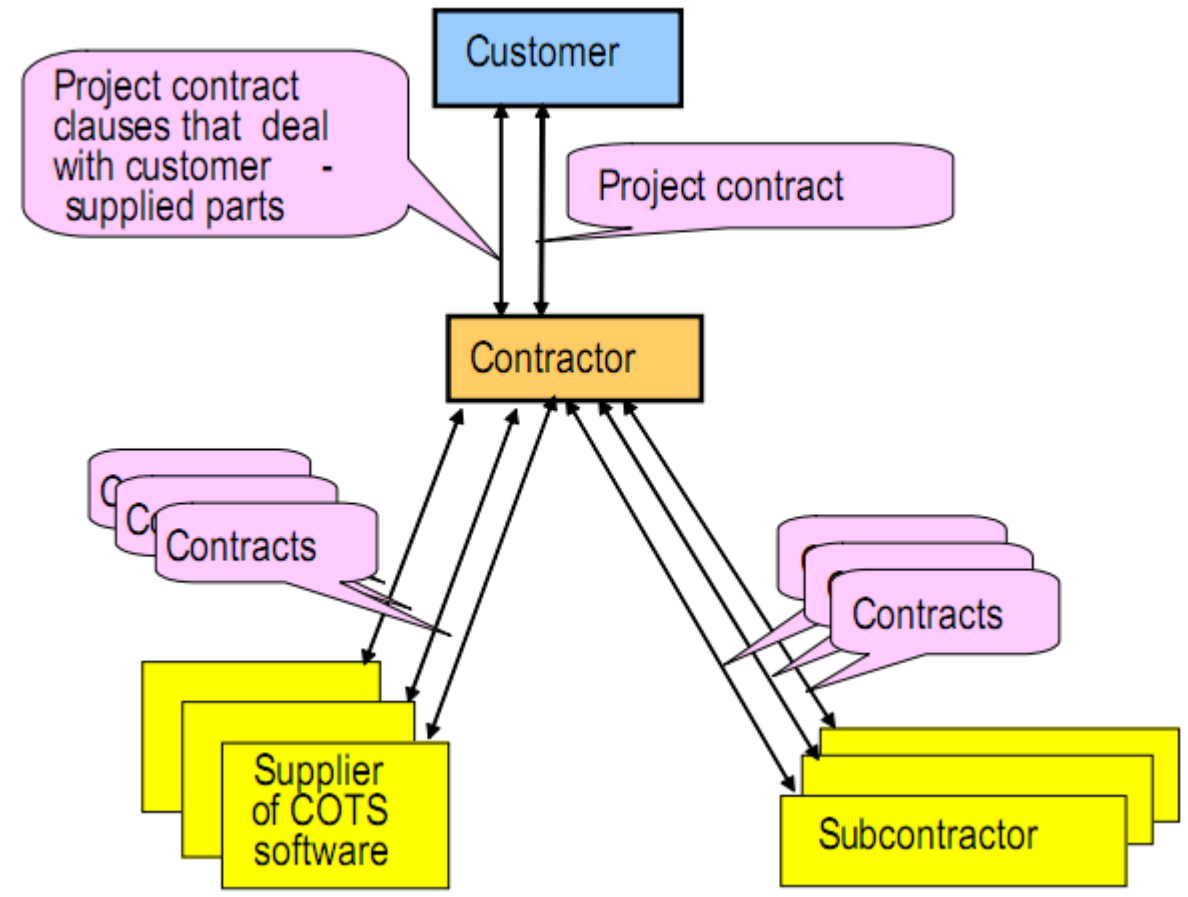
5. External participants
Types

- Subcontractors
  - currently called “outsourcing” organizations
  - benefits: staff availability, special expertise or low prices
- Suppliers of COTS software and reused software modules
  - benefits: reduce time and cost, increase quality
- The customer themselves
  - to apply the customers’ special expertises

5. External participants
Typical contracting structures



simple structure



5. External participants Benefits

- For the contractor
  - **budget reductions**
  - **shorter** project **schedule**
  - overcoming **shortages of professional staff**
  - **expertise acquired** in areas that need specialisation
- For the customer
  - project **cost reductions**
  - **protection** of commercial secrets
  - **provision of employment** to internal software development department

5. External participants Risks

- **Delayed** completion of the project parts
  - when external participants parts are late → whole project delayed
- **Low quality** of parts
  - (a) more defects or more severe
  - (b) non-standard coding of documentation → cause difficulty in testing and in maintenance
- Increased probability of **difficulties in maintenance**
- **Loss of control** over development
  - communication with external participants' teams may be interrupted → prevents assessment of the project's progress

5. External participants Assuring the quality

- Objectives:
  - Prevent late completion
  - Assure the quality of work
  - Assure enough documentation
  - Assure complete control

5. External participants SQA tools utilised

- Requirements documents reviews
  - assures a list of the requirements is correct and complete - reduces delays and low quality
- Choice of external participant
- Project co-ordination and joint control committee
- Participation in design reviews
  - contractor acts as a full member of the review
- Participation in software testing
  - design reviews of the planning and design of the tests, reviews of the test results, follow-up meetings for the corrections and regression testing

5. External participants
SQA tools utilised (cont.)

- Certification of external participants' team leaders and other staff
  - to ensure an professional capacities of project teams - minimise risk of low quality
- Progress reports
  - the follow-up of the usage of resources
  - the follow-up of the project budget
- Review of deliverables and acceptance tests

Câu hỏi hiểu bài
(SV xem lại bài học, cô sẽ gọi trả lời)

1. **Liệt kê các thành phần của hệ thống đảm bảo chất lượng phần mềm.**
2. **Nêu các thành phần cơ bản (cơ sở hạ tầng) của chất lượng phần mềm.**
3. **Giải thích 2 nguyên tắc kiểm thử phần mềm: "Defect clustering", "The pesticide paradox"**
4. **Quy trình kiểm thử phần mềm?**
5. **Có những mức kiểm thử nào? Phân biệt các mức này qua: mục đích (verification, validation), thời điểm thực hiện, người thực hiện**
6. **Kiểm thử phi chức năng (non-functional test) là gì? Cho ví dụ ít nhất 2 loại thuộc kiểm thử này.**
7. **Phân biệt kiểm thử xác nhận (re-testing) và kiểm thử hồi quy (regression testing).**
8. **Review là gì? Có thể review những gì? Phân biệt walkthrough và inspection.**
9. Có các loại bảo trì phần mềm nào? Làm thế nào để việc bảo trì được thuận lợi.
10. Các câu hỏi phần Kiểm tra hiểu bài về Quy trình kiểm thử (slide 42)