SOFTWARE TESTING
(DYNAMIC VERIFICATION)

Week 8

Software Quality

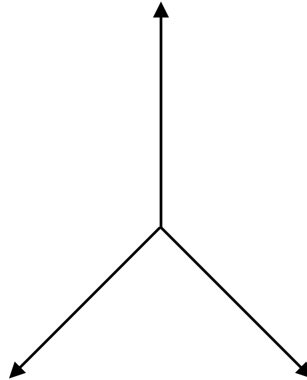
- *Standard Glossary of SW Engineering Terminology* [IEEE610.12]:
  Quality: (1) The degree to which a system, component, or process *meets **specified requirements***. (2) The degree to which a system, component, or process *meets customer or **user needs or expectations***.

Software Quality

- ***Software Quality***: Conformance to
  - ***explicitly stated*** *functional and non-functional (performance, etc) requirements*,
  - ***explicitly documented*** *development standards*, and
  - *implicit characteristics* that are expected of all ***professionally*** *developed software*.

Dimensions of SW Quality

Dimension: ***Adaptability to new environments***
Sys. Prop.: *Portability, Reusability, Interoperability*



Dimension: ***Ability to undergo change***
Sys. Prop.: *Maintainability, Testability, Flexibility*

Dimension: **Operational Characteristics**
Sys. Prop.: *Correctness, Efficiency, Reliability, Integrity, Usability*

System Properties

- ***Portability***: The effort required to transfer the system from one hardware and/or software environment to another.
- ***Reusability***: The extent to which the system (or part of it) can be reused in other applications.
- ***Interoperability***: The effort required to couple the system to another.
- ***Maintainability***: The effort required to introduce a modification (usually a correction) into the system.
- ***Flexibility***: The effort required to modify or customize the system in operation.
- ***Testability***: The effort required to test the system to ensure that it performs its intended function.

System Properties

- ***Correctness***: The extent to which the system satisfies its specification and fulfills the users' needs.
- ***Reliability***: The extent to which the system can be expected to perform its intended function with required precision and without failure.
- ***Efficiency***: The amount of computing resources (space, time) required by the system to perform its function.
- ***Integrity***: The extent to which access to the system or its data by unauthorized persons can be controlled.
- ***Usability***: The effort required to learn, operate, prepare input and interpret output of the system.

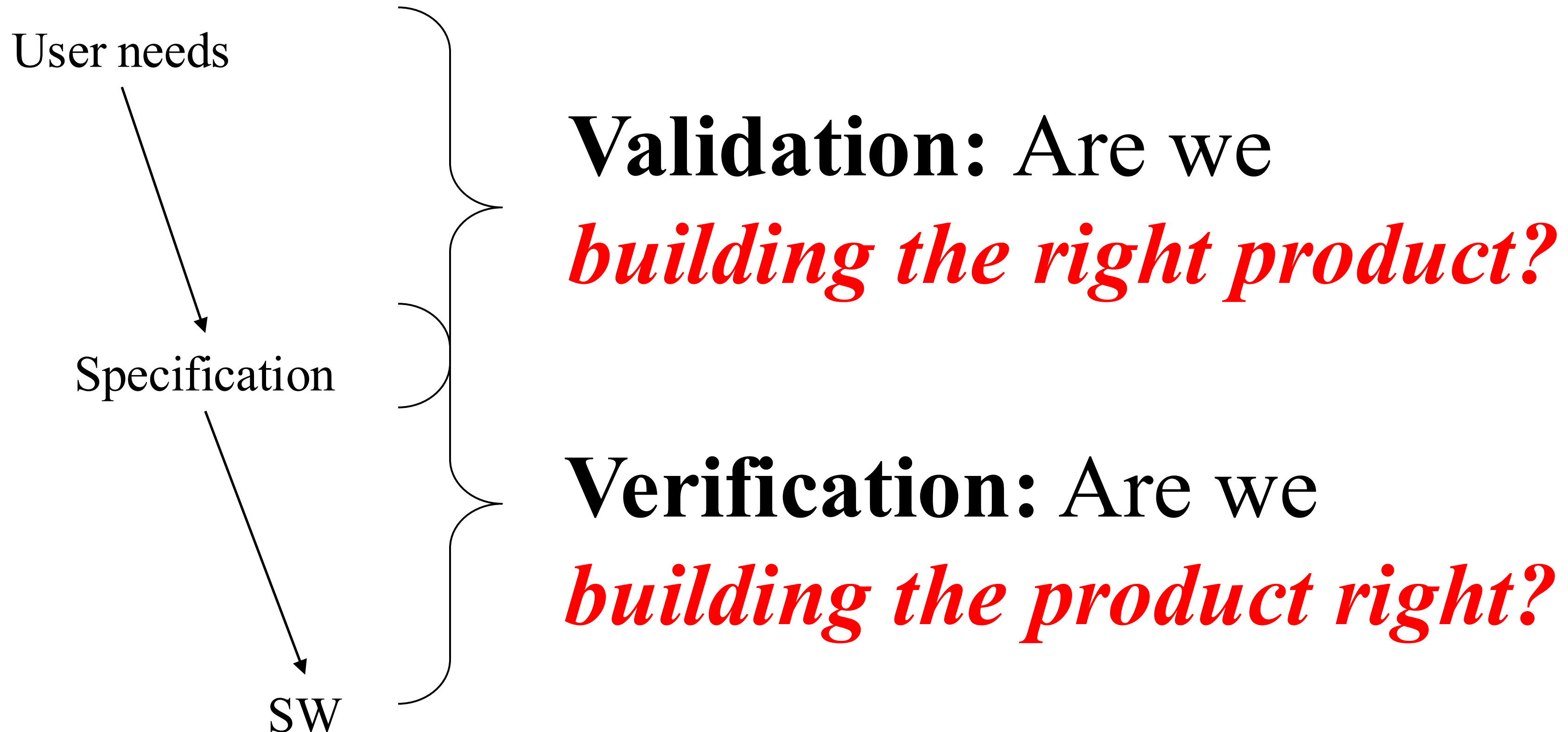
Software Quality

- *Standard Glossary of SW Engineering Terminology* [IEEE610.12]:
  Quality assurance: (1) A planned and systematic pattern of all actions necessary to provide *adequate confidence* that an item or product conforms to established technical requirements. (2) A set of activities designed to *evaluate the process* by which products are developed or manufactured.

Quality Assurance

- application of *sound* technical methods and tools
- formal technical reviews and inspections
- ***software testing***
- enforcement of *standards*
- Documentation – is the process/product well guided ???
- control of change – is the product maintainable ???
- extensive measurement -
- record keeping and reporting of the process

Verification & Validation



Principles of SW Testing

1. *Testing* is a process of *executing a program with the intent of finding a defect*. So, there must be some program code to be executed.
2. A *good test case* is one that has a high probability of *finding an as yet undiscovered defect*. So, the test cases (the program input) should be selected systematically and with care, both for correct and incorrect behavior.
3. A *successful test is one that uncovers an as yet undiscovered defect*. So, testing is psychologically ***destructive*** since it tries to demolish the software that has been constructed.

Principles of SW Testing

4. Testing *cannot* show the absence of defects, it can only show that they are present (Dijkstra). [Testing is not *formal verification*.]
5. Testing is quite an ineffective method of quality assurance. [Though, usually the most applicable one.]
6. Successful testing shall be *followed by* a separate *debugging* phase.
7. Testing is also by itself a *process* that must be systematically managed (and assisted with special testing tools).

Jargon of Testing

- **Error (yanlışlık):** A mistake (*human* action) made by a software developer. It might be a total misinterpretation of user requirements, or a simple typograhpical misprint. An error introduces a defect into the *software code*.
- **Defect, fault, bug (arıza, sorun):** A difference between the incorrect program and its correct version; a coding error. A defect in the software, if encountered during *execution*, *may cause a failure*.
- **Failure (başarısızlık)**: An externally observable deviation of the functional software from its specification; an *incorrect result of computation*.

Facts of Testing

- *V&V engineer must define "correct" and "incorrect."*
- There must be *a specification against which to check the results of testing*.
- Full automation of testing is impossible: (4 reasons)
  1. in theory, the *total behavior of a program is undecidable* (halting, failures);
  2. in practice, *exhaustive testing is intractable*;
  3. *tracking of (technical) failures to (human) errors is impossible*;
  4. we can never be sure that the testing tool (a program) works correctly.

Why testing? Why *systematic* testing?

- According to several empirical studies, a (*professionally produced commercial*) SW system...
  - ...contains ***3 – 30 defects per 1000 lines of code***
  - ..., the average debugging effort is ***12 hours of working time per defect***
  - ..., ***maintenance*** is about *half of software development costs*, mostly in error removal

An Example

- The program reads three integer values. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene, isosceles, or equilateral. *Write test cases* (*specific input values*) *that you feel would adequately test this program.*
- In a valid triangle, no side may have a length of zero or less, and each side must be shorter than the sum of all sides divided by 2.
- ***Equilateral*** (eşkenar) triangle: all sides are of equal length.
- ***Isosceles*** (ikizkenar) triangle: two sides are of equal length.
- ***Scalene*** (çeşitkenar) triangle: all sides are of unequal length.

Example... cont'd

- In mathematics, the number of integer values is infinite. However, computers have finite space which limits the number of values that can be processed. Let us assume that our triangle program is running in a tiny computer with 10.000 as the largest integer value. Then there are $10^4 * 10^4 * 10^4 = 10^{12}$ possible length combinations of triangle sides (including the invalid ones).
- Suppose you are a very fast tester, running and checking 1000 tests per second, 24 hours per day, 365 days per year.
- Then the exhaustive testing effort (testing each possible length combination) would take over 31.7 years.

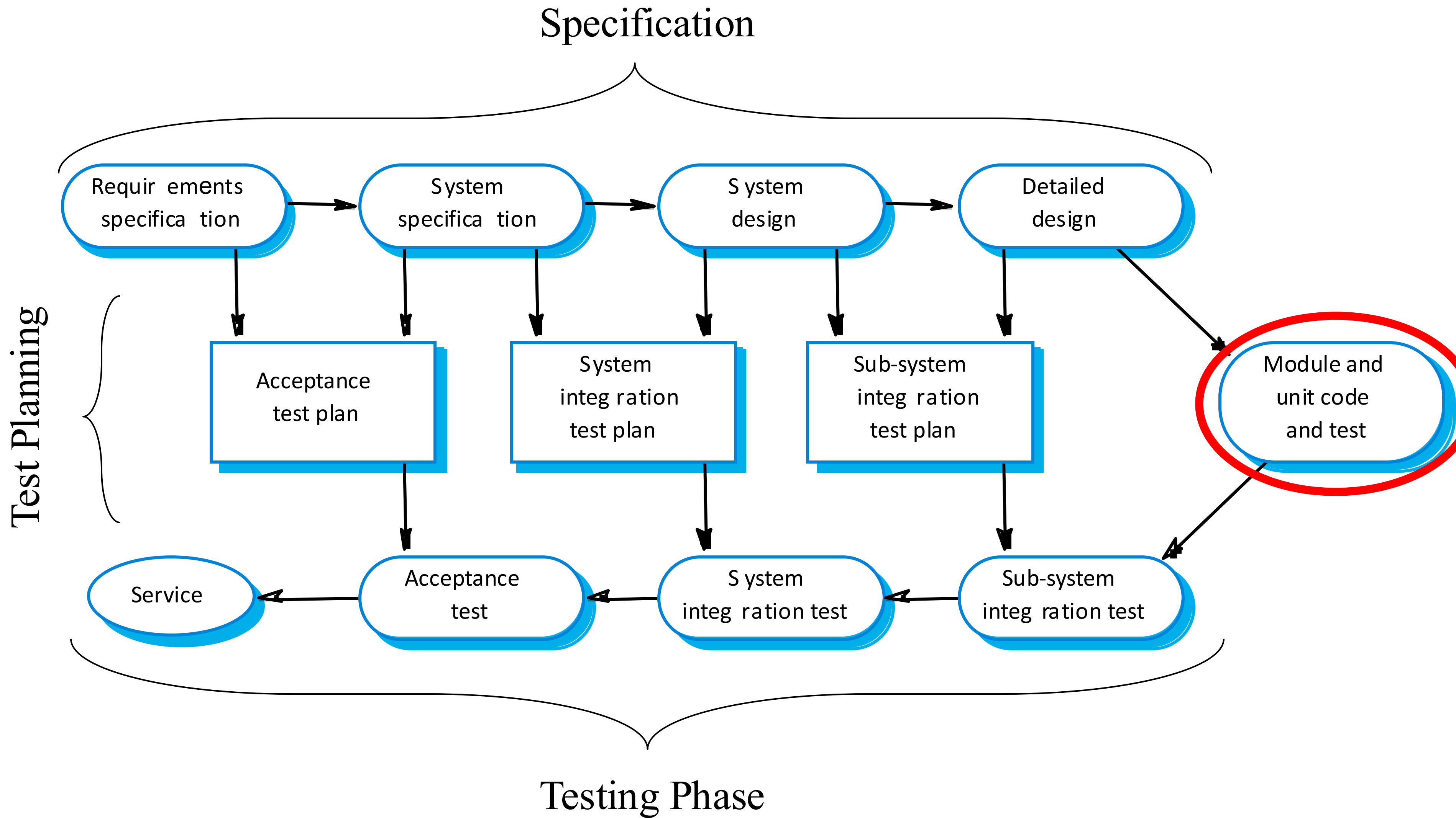
Some of Possible Test Cases [1]

1. (5, 3, 4): scalene - *çeşitkenar*
2. (3, 3, 4): isosceles - *ikizkenar*
3. (3, 3, 3): equilateral - *eşkenar*
4. (50, 50, 25): isosceles
5. (25, 50, 50): isosceles (permutation)
6. (50, 25, 50): isosceles (permutation)
7. (10, 10, 0): invalid (zero)
8. (3, 3, -4): invalid (negative)
9. (5, 5, 10): invalid (too long)
10. (10, 5, 5): invalid (too long, perm.)
11. (5, 10, 5): invalid (too long, perm.)
12. (8, 2, 5): invalid (Too long)
13. (2, 5, 8): invalid (Too long, perm.)
14. (2, 8, 5): invalid (Too long, perm.)
15. (8, 5, 2): invalid (Too long, perm.)
16. (5, 8, 2): invalid (Too long, perm.)
17. (5, 2, 8): invalid (Too long, perm.)
18. (0, 0, 0): invalid (all zeros)
19. (@, 4, 5): invalid (non-integer)
20. (3, $, 5): invalid (non-integer)
21. (3, 4, %): invalid (non-integer)
22. (, 4, 5): invalid (missing input)
23. (3,,5): invalid (missing input)
24. (3, 4,): invalid (missing input)

Some Remarks

- *most test cases represent invalid inputs*
- *each valid triangle type is tested at least once*
- *permutations* are used to check that *the order of the input values does not affect the result*
- *boundary input values* are used (e.g., length of exactly zero, length of exactly the sum of all sides divided by 2) are *more likely points of faults*.
- *input values of wrong type* (e.g., non-integers) are *used*
- the *number of test cases is much smaller than the number of all possible inputs*

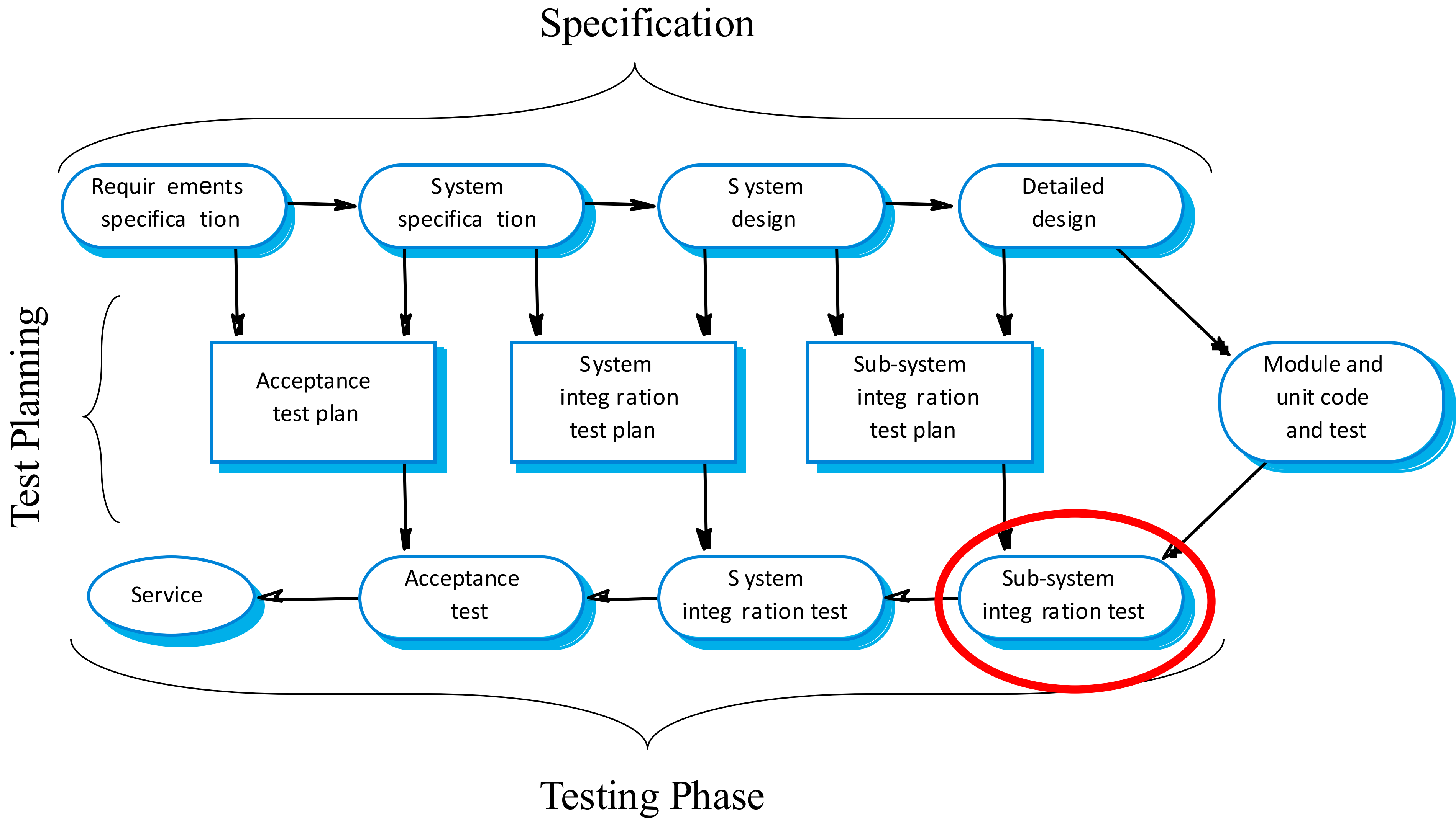
The V-Model of SW Development



Module (unit) testing:

- each independent unit tested separately
- **level**: *source code*
- *need for simulated execution environment*

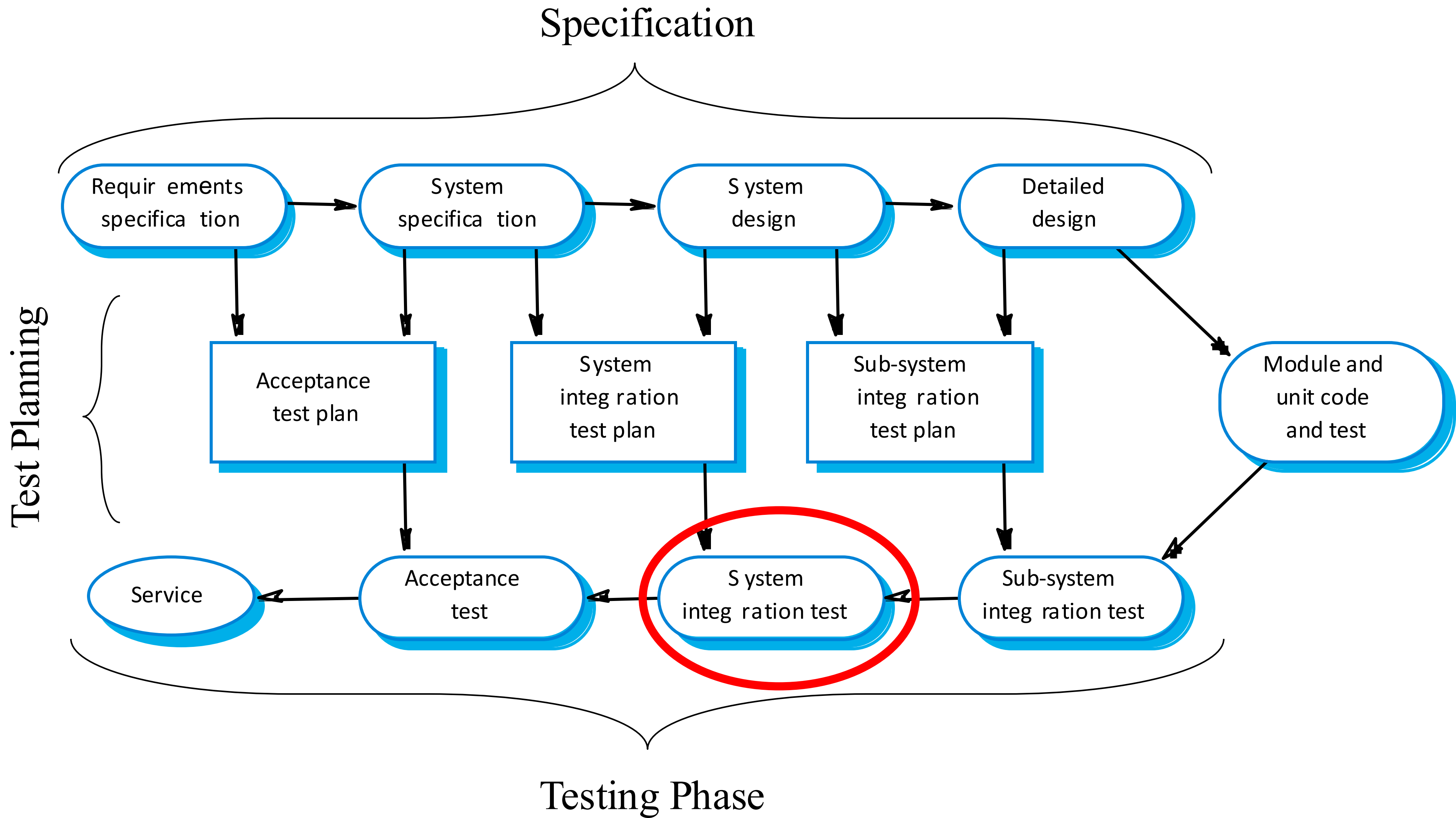
The V-Model of SW Development



Integration testing:

- *modules grouped into subsystems* for testing
- *"big bang:"* all the modules tested as a whole
- *incremental approaches: modules into subsystems*
- **level-wise** ("stubs" and "drivers")
- **level**: interfaces between modules

The V-Model of SW Development



System Testing

- the *whole system* (including hardware, databases, sensors, …) *tested*
- **target**: performance, capacity, fault-tolerance, security, configuration, … (non-functional req.s)
- **level**: external (i.e., user) interface

Special Types of Testing

- Volume testing
- Load / stress testing
- Security testing
- Performance testing
- Configuration testing
- Installability testing
- Recovery testing
- Reliability / availability testing
- Maintainability testing
- Protocol conformance testing, etc.

Volume Testing

- a *non-functional* test.
- refers to *testing a software application with a certain amount of data*
  - the ***database size*** or
  - the size of an interface file that is the subject of volume testing.
- Examples (for *performance* testing)
  - volume testing an application with a specific database size,
    - expand your database to that size and then test the application's ***performance*** on it.
  - suppose a requirement for your application is to interact with an interface file (could be any file such as .dat, .xml);
  - this interaction could be reading and/or writing on to/from the file. You will create a sample file of the size you want and then test the application's functionality with that file in order to test the performance.

Security Testing (ST)

- determines that a SW product protects data and maintains functionality as intended.
- *six basic concepts* to be covered by ST
  - *Confidentiality*: protection against disclosure of info to unauthorized parties
  - *Integrity*: a measure that allows the receiver to determine that information provided by the system is correct.
  - *Authentication*: confirmation identity of a person, tracing origins of an artifact or ensuring that a product is what its packaging claims it is, or making sure a computer program is a trusted one.
  - *Authorization*: determining that a requester is allowed to receive a service or perform an operation.
  - *Availability*: Assuring that relevant service will be ready for use when expected
  - *Non-repudiation*: assuring that some message transferred is sent and received by the parties claiming to have sent and received it.

Performance Testing

- Determines how a system performs in terms of *responsiveness* and *stability* under a particular workload.
- Also investigates, measures, validates or verifies *other system properties such as scalability, reliability and resource usage*.

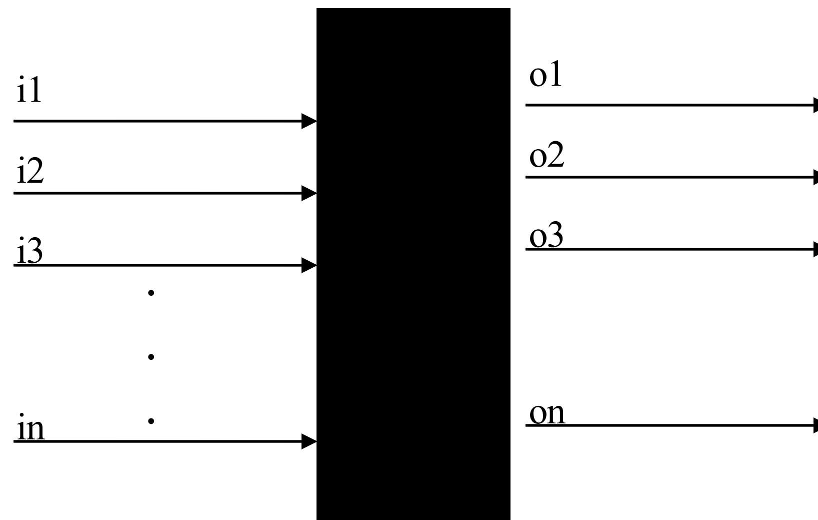
Acceptance Testing

- user involvement (alpha, beta)
- "actual needs" tested
- ***usability testing*** at the user interface
- development team in development environment:
  - "standard", general usability errors
- real user representatives in laboratory environment:
  - task-specific usability problems (real tasks, talk-aloud, taping, post-analysis by experts)

Black-box (functional) testing

- *internal details* of modules or subsystems are *hidden* and cannot be studied from outside
- concentrates on the *interfaces* of *modules and (sub)systems* (e.g. user interface)
- externally observable functionality and input-output behavior
- based on input *classification*
- especially suitable for *integration*, *system*, and *acceptance testing*

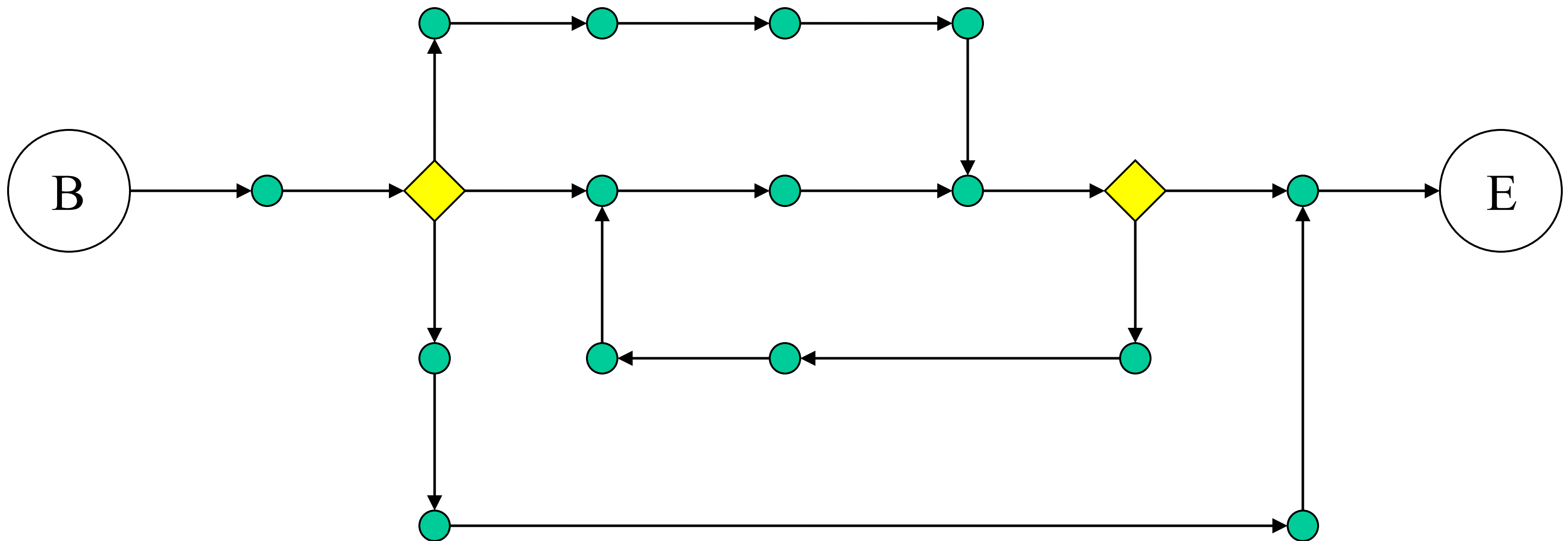
Black-box (functional) testing



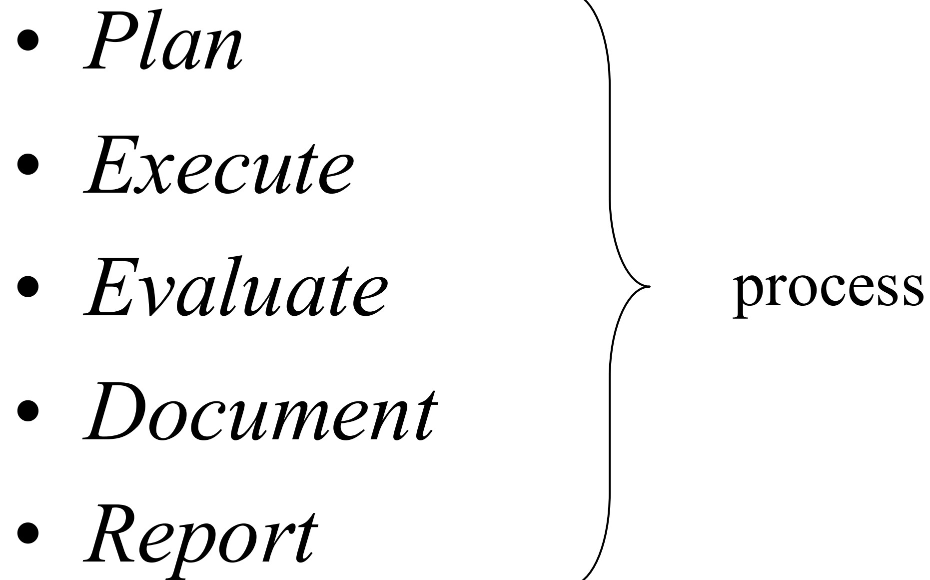
White-box (structural) testing

- *structure of software examined* in detail at the level of program code
- *objective to traverse as many paths over the code* as considered necessary
- *based on control flow and data flow*
- *several forms of coverage* (path, statement, branch, …)
- especially *suitable for module (unit) testing*

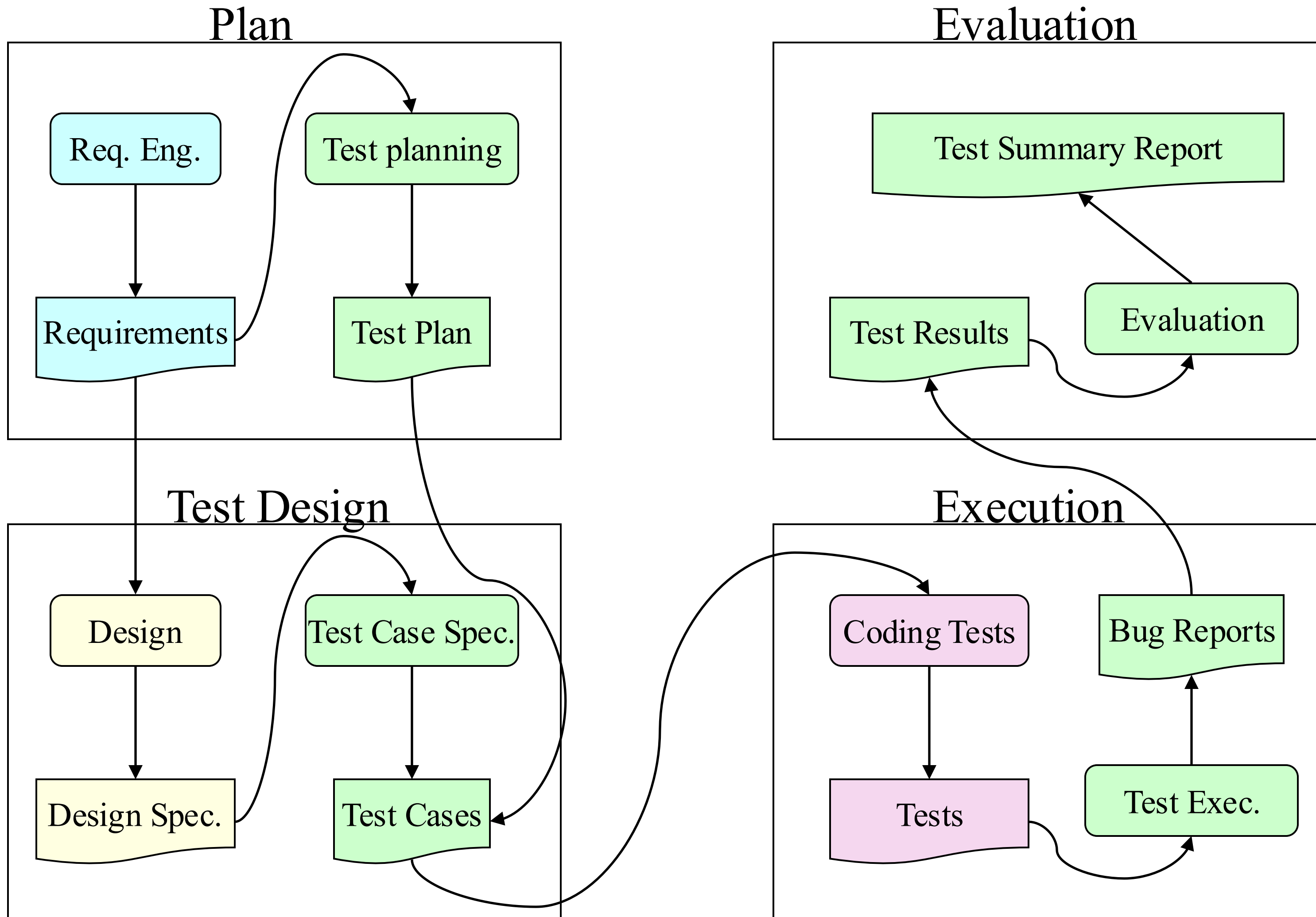
White-box (structural) testing



Management of Testing



Testing Process



Standard for Software Test Documentation [IEEE 829]

1) *Test plan*: the scope, approach, resources, and schedule of the testing activities.
2) *Test-design specification*: the refinements of the test approach, and the features to be tested by the design and its associated tests.
3) *Test-case specification*: a test case identified by a test-design specification.
4) *Test-procedure specification*: the steps for executing a set of test cases or, more generally, the steps used to analyze a software item in order to evaluate a set of features.

Standard for Software Test Documentation [IEEE 829]

5) *Test-item transmittal report*: the test items being transmitted for testing, including the person responsible for each item, its physical location, and its status.
6) *Test log*: a chronological record of relevant details about the execution of tests.
7) *Test-incident report* (***bug report***)**:** any event that occurs during the testing process which requires investigation.
8) *Test-summary report*: the results of the designated activities, and evaluations based on these results.

Test plan

1) *Test-plan identifier*: specifies the unique identifier assigned to the test plan.
2) *Introduction*: summarizes the software items and software features to be tested, provides references to the documents relevant for testing (overall project plan, quality assurance plan, configuration management plan, applicable standards…).
3) *Test items*: identifies the items to be tested, including their version/revision level; provides references to the relevant item documentation (requirements specification, design specification, user's guide, operations guide, installation guide, …); also identifies items which are specifically excluded from testing.

Test Plan

4) *Features to be tested*: identifies all software features and their combinations to be tested, identifies the test design specification associated with each feature and each combination of features.
5) *Features not to be tested*: identifies all features and significant combinations of features which will not be tested, and the reasons for this.
6) *Approach*: describes the overall approach to testing (the testing activities and techniques applied, the testing of non-functional requirements such as performance and security, the tools used in testing); specifies completion criteria (for example, error frequency or code coverage); identifies significant constraints such as testing-resource availability and strict deadlines; serves for estimating the testing efforts.

Test Plan

7) *Item pass/fail criteria*: specifies the criteria to be used to determine whether each test item has passed or failed testing.
8) *Suspension criteria and resumption*: specifies the criteria used to suspend all or portion of the testing activity on the test items (at the end of working day, due to hardware failure or other external exception, …), specifies the testing activities which must be repeated when testing is resumed.
9) *Test deliverables*: identifies the deliverable documents, typically test-design specifications, test-case specifications, test-procedure specifications, test-item transmittal reports, test logs, test-incident reports, description of test-input data and test-output data, description of test tools.

Test Plan

10) *Testing tasks*: identifies the set of tasks necessary to prepare and perform testing (description of the main phases in the testing process, design of verification mechanisms, plan for maintenance of the testing environment, …).

11) *Environmental needs*: specifies both the necessary and desired properties of the test environment (hardware, communications and systems software, software libraries, test support tools, level of security for the test facilities, drivers and stubs to be implemented, office or laboratory space, …).

Test Plan

12) *Responsibilities*: identifies the groups of persons responsible for managing, designing, preparing, executing, witnessing, checking, and resolving the testing process; identifies the groups responsible for providing the test items (section 3) and the environmental needs (section 11).

13) *Staffing and training needs*: specifies the number of testers by skill level, and identifies training options for providing necessary skills.

14) *Schedule*: includes test milestones(those defined in the overall project plan as well as those identified as internal ones in the testing process), estimates the time required to do each testing task, identifies the temporal dependencies between testing tasks, specifies the schedule over calendar time for each task and milestone.

Test Plan

15) Risks and contingencies: identifies the high-risk assumptions of the test plan (lack of skilled personnel, possible technical problems, …), specifies contingency plans for each risk (employment of additional testers, increase of night shift, exclusion of some tests of minor importance, …).

16) Approvals: specifies the persons who must approve this plan.

Test-case specification

1) *Test-case-specification identifier*: specifies the unique identifier assigned to this test-case specification.
2) *Test items*: identifies and briefly describes the items and features to be exercised by this test case, supplies references to the relevant item documentation (requirements specification, design specification, user's guide, operations guide, installation guide, …).
3) *Input specifications*: specifies each input required to execute the test case (by value with tolerances or by name); identifies all appropriate databases, files, terminal messages, memory resident areas, and external values passed by the operating system; specifies all required relationships between inputs (for example, timing).

Test-case specification

4) *Output specifications*: specifies all of the outputs and features (for example, response time) required of the test items, provides the exact value (with tolerances where appropriate) for each required output or feature.
5) *Environmental needs*: specifies the hardware and software configuration needed to execute this test case, as well as other requirements (such as specially trained operators or testers).
6) *Special procedural requirements*: describes any special constraints on the test procedures which execute this test case (special set-up, operator intervention, …).
7) *Intercase dependencies*: lists the identifiers of test cases which must be executed prior to this test case, describes the nature of the dependencies.

Test-incident report (bug report)

1) *Bug-report identifier:* specifies the unique identifier assigned to this report.
2) *Summary:* summarizes the (bug) incident by identifying the test items involved (with version/revision level) and by referencing the relevant documents (test procedure specification, test-case specification, test log).

Test-incident report (bug report)

3) *Bug description*: provides a description of the incident, so as to correct the bug, repeat the incident or analyze it off-line:
   a. Inputs.
   b. Expected results.
   c. Actual results.
   d. Date and time.
   e. Test-procedure step.
   f. Environment.
   g. Repeatability (whether repeated; whether occurring always, occasionally or just once).
   h. Testers.
   i. Other observers.
   j. Additional information that may help to isolate and correct the cause of the incident; for example, the sequence of operational steps or history of user-interface commands that lead to the (bug) incident.
4) *Impact*: Priority of solving the incident / correcting the bug (urgent, high, medium, low).

Test-summary report

1) *Test-summary-report identifier*: specifies the unique identifier assigned to this report.
2) *Summary*: summarizes the evaluation of the test items, identifies the items tested (including their version/revision level), indicates the environment in which the testing activities took place, supplies references to the documentation over the testing process (test plan, test-design specifications, test-procedure specifications, test-item transmittal reports, test logs, test-incident reports, …).

Test-summary report

3) *Variances*: reports any variances/deviations of the test items from their design specifications, indicates any variances of the actual testing process from the test plan or test procedures, specifies the reason for each variance.
4) *Comprehensiveness assessment*: evaluates the comprehensiveness of the actual testing process against the criteria specified in the test plan, identifies features or feature combinations which were not sufficiently tested and explains the reasons for omission.
5) *Summary of results*: summarizes the success of testing (such as coverage), identifies all resolved and unresolved incidents.

Test-summary report

6) *Evaluation*: provides an overall evaluation of each test item including its limitations (based upon the test results and the item-level pass/fail criteria).
7) *Summary of activities*: summarizes the major testing activities and events, summarizes resource consumption (total staffing level, total person-hours, total machine time, total elapsed time used for each of the major testing activities, …).
8) *Approvals*: specifies the persons who must approve this report (and the whole testing phase).

Inspection checklist for test plans:

1) Have all materials required for a test plan inspection been received?
2) Are all materials in the proper physical format?
3) Have all test plan standards been followed?
4) Has the testing environment been completely specified?
5) Have all resources been considered, both human and hardware/software?
6) Have all testing dependencies been addressed (driver function, hardware, etc.)?
7) Is the test plan complete, i.e., does it verify all of the requirements? *For unit testing*: does the plan test all functional and structural variations from the high-level and detailed design?)
8) Is each script detailed and specific enough to provide the basis for test case generation?
9) Are all test entrance and exit criteria sufficient and realistic?
10) Are invalid as well as valid input conditions tested?
11) Have all pass/fail criteria been defined?

Inspection checklist for test plans:

12) Does the test plan outline the levels of acceptability for pass/fail and exit criteria (e.g., defect tolerance)?
13) Have all suspension criteria and resumption requirements been identified?
14) Are all items excluded from testing documented as such?
15) Have all test deliverables been defined?
16) Will software development changes invalidate the plan? (*Relevant for unit test plans only*.)
17) Is the intent of the test plan to show the presence of failures and not merely the absence of failures?
18) Is the test plan complete, correct, and unambiguous?
19) Are there holes in the plan; is there overlap in the plan?
20) Does the test plan offer a measure of test completeness and test reliability to be sought?
21) Are the test strategy and philosophy feasible?

Inspection checklist for test cases:

1) Have all materials required for a test case inspection been received?
2) Are all materials in the proper physical format?
3) Have all test case standards been followed?
4) Are the functional variations exercised by each test case required by the test plan? (*Relevant for unit test case documents only*.)
5) Are the functional variations exercised by each test case clearly documented in the test case description? (*Relevant for unit test case documents only*.)
6) Does each test case include a complete description of the expected input, and output or result?
7) Have all testing execution procedures been defined and documented?
8) Have all testing dependencies been addressed (driver function, hardware, etc.)?
9) Do the test cases accurately implement the test plan?

References

[1] Myers, *The Art of Software Testing*, 1978

Appendix A: Drivers and Stubs

- ***Idea***: develop and test software in "pieces" (modular approach)
- ***Problem***: how to test a "piece" if the other "pieces" that it uses have not yet been developed (and vice versa).
- ***Solution***: stubs and drivers.

- ***Relevance***: White-box testing
  - must run the code with predetermined input and check to make sure that the code produces predetermined outputs.
  - Often testers write stubs and drivers for white-box testing.

Appendix A: Drivers

Test Drivers:

Driver is a the piece of code that passes test cases to another piece of code. Test Harness or a test driver is supporting code and data used to provide an environment for testing part of a system in isolation. It may be defined as a software module used to invoke a module under test and provide test inputs, control and, monitor execution, and report test results or ***most simplistically a line of code that calls a method and passes that method a value.***

For example, if you wanted to move a fighter on the game, the driver code would be
moveFighter(Fighter, LocationX, LocationY);

This driver code would likely be called from the main method. A white-box test case would execute this driver line of code and check "fighter.getPosition()" to make sure the player is now on the expected cell on the board.

Appendix A: Stubs

A Stub is a dummy procedure, module or unit that stands in for an unfinished portion of a system.

Four basic types of Stubs for Top-Down Testing are:

1 Display a trace message
2 Display parameter value(s)
3 Return a value from a table
4 Return table value selected by parameter

A stub is a computer program used as a substitute for the body of a software module that is or will be defined elsewhere or a dummy component or object used to simulate the behavior of a real component until that component has been developed.

Ultimately, the dummy method would be completed with the proper program logic. However, developing the stub allows the programmer to call a method in the code being developed, even if the method does not yet have the desired behavior.

Stubs and drivers are often viewed as throwaway code. However, they do not have to be thrown away: Stubs can be "filled in" to form the actual method. Drivers can become automated test cases.

Integrity

- ... are addressed in processes to source and create SW components and deliver them to customers
  - ... contain controls to enhance confidence that SW was not modified without consent of supplier.