BLG 475E: Software Quality and Testing
Fall 2014-15

Assist. Prof. Ayşe Tosun Mısırlı
tosunmisirli@itu.edu.tr

İstanbul Teknik Üniversitesi

Course plan

- Fundamentals of Testing (11th Sept by Turkcell)
- Testing strategies
- Testing types (Turkcell)
- Testing in SDLC (Turkcell)
- Processes
- Introduction to software quality
- Software quality models, software quality assurance
- Software defects and SQA activities
- Software quality metrics
- Data flow models and their roles in testing
- Test automation and performance tools (Turkcell)
- Test deployment and Release management (Turkcell)
- Test challenge (Turkcell)

How to pass the course?

- Rule #1: Interactive discussion during the class (Criticize and discuss!)
- Rule #2: Regular attendance to the class
- Other conditions:
  - Midterm (8th week)
  - Small exercises during class
  - Assignments
  - Final exam

Outline

- Software testing
  - Definitions
  - Objectives
  - Strategies
  - Types

Software testing

- 1st Software Quality Assurance Tool
  - Testing was confined to the final stage of the development, after the entire package had been completed.
- Other early SQA tools
  - Walkthroughs, code reviews, unit testing, etc.
- Testing is the most time consuming and expensive SQA.

Cost of software testing

- A survey in 1994 (by Perry)
  - 24% of project development budget is allocated to testing.
  - 32% of project management budget is allocated to testing activities.
  - Time resources
    - 27% of project time is spent to testing.
  - Participants
    - Allocated time to testing is around 45%
    - But pressure towards the release time force managers reduce scheduled testing time!
- According to Brooks
  - 50% of development costs is in unit and system testing.

Definitions

- "Testing is the process of executing a program with intention of finding errors." (Myers, 1979)

- "(1) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.

  (2) The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item." (IEEE Std.610.12, 1990)

Definition by Galin 2004

- Software testing is a **formal** process carried out by a **specialized testing team** in which a software unit, several integrated software units or an entire software package are examined by **running the programs on a computer**. All the associated tests are performed according to **approved test procedures** on **approved test cases**.

Definition: Key Characteristics

- **Formal**
  - Software test plans (not *ad-hoc* examination)
- **Specialized testing team**
  - Independent team or external consultants specialized in testing to eliminate bias and to guarantee effective testing
- **Running the programs**
  - QA activities like code reviews (that does not involve program execution) are not considered as testing.
- **Approved test procedures**
  - Testing procedures approved as confirming to SQA procedures adopted by the developing organization.
- **Approved test cases**
  - Test cases should be approved before testing begins. No editions are allowed.

Objectives of software testing

- **Direct**
  - To identify and reveal as many errors as possible
  - To bring the tested software, after correction of the identified defects and retesting, to an acceptable level of quality.
  - To perform the required tests efficiently and effectively, within the limits of budgets and scheduling.
- **Indirect**
  - To compile a record of software errors for use in error prevention (by corrective and preventive actions)

"If your goal is to show the absence of errors you won't discover many. If your goal is to show the presence of errors, you will discover a large percentage of them." (Myers 1979)

Software test strategies

- Big bang testing
  - Testing in its entirety
- Incremental testing
  - Testing modules as they are completed (unit tests); then testing groups of *tested* modules integrated with newly completed modules (integration tests)
  - Top-down OR Bottom-up
  - Stubs and drivers

Incremental: Top-down

Stage 1: Unit tests of module 11.

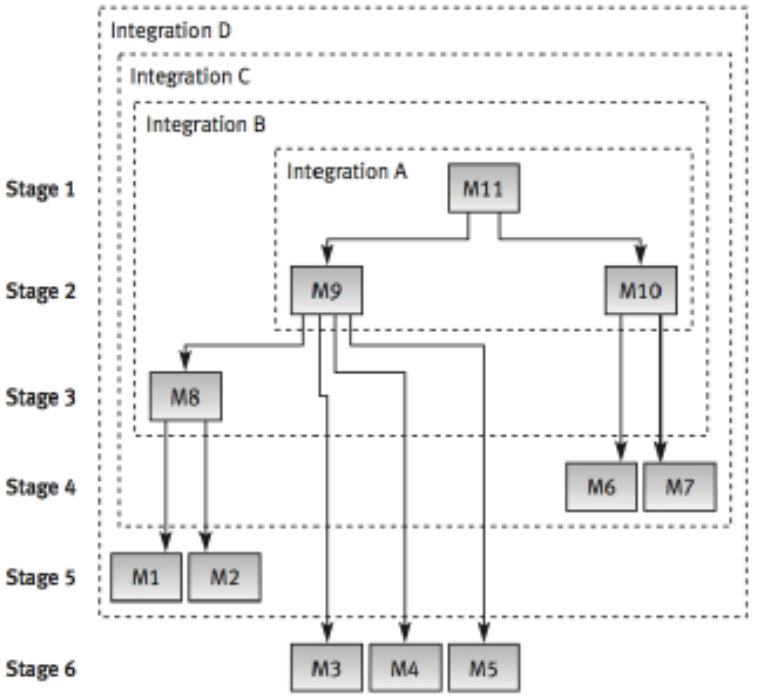
Stage 2: Integration test A of module 11 integrated with modules 9 and 10, developed in the current stage.

Stage 3: Integration test B of A integrated with module 8, developed in the current stage.

Stage 4: Integration test C of B integrated with modules 6 and 7, developed in the current stage.

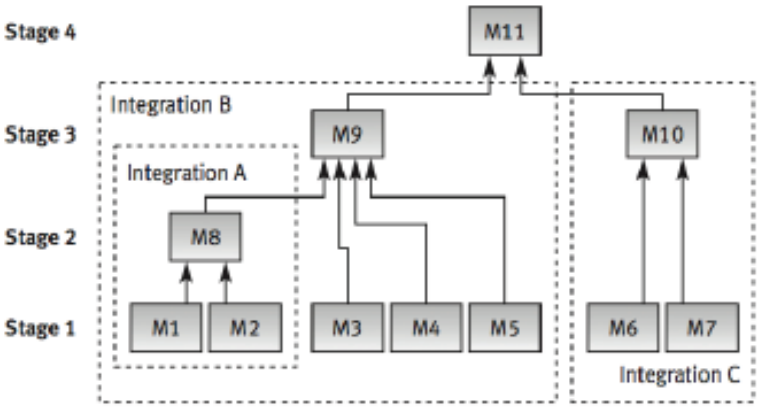
Stage 5: Integration test D of C integrated with modules 1 and 2, developed in the current stage.

Stage 6: System test of D integrated with modules 3, 4 and 5, developed in the current stage.



Galin D., "Software Quality Assurance: From theory to implementation", Addison-Wesley, 2004

Incremental: Bottom-up

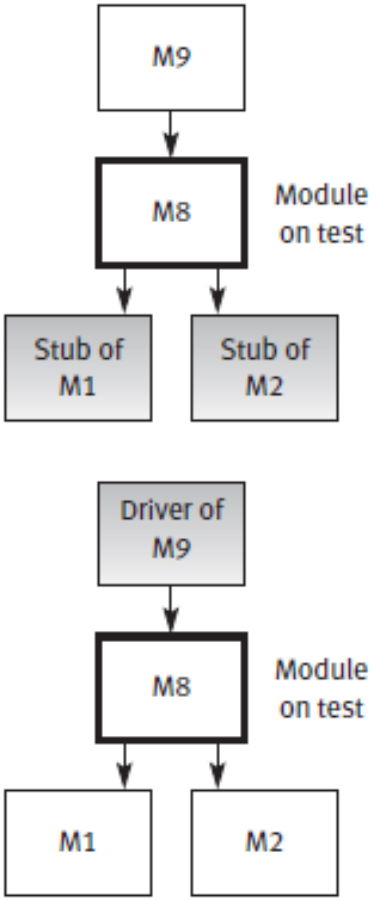


- Stage 1: Unit tests of modules 1 to 7.
- Stage 2: Integration test A of modules 1 and 2, developed and tested in stage 1, and integrated with module 8, developed in the current stage.
- Stage 3: Two separate integration tests, B, on modules 3, 4, 5 and 8, integrated with module 9, and C, for modules 6 and 7, integrated with module 10.
- Stage 4: System test is performed after B and C have been integrated with module 11, developed in the current stage.

Galin D., "Software Quality Assurance: From theory to implementation", Addison-Wesley, 2004

Incremental: Stubs

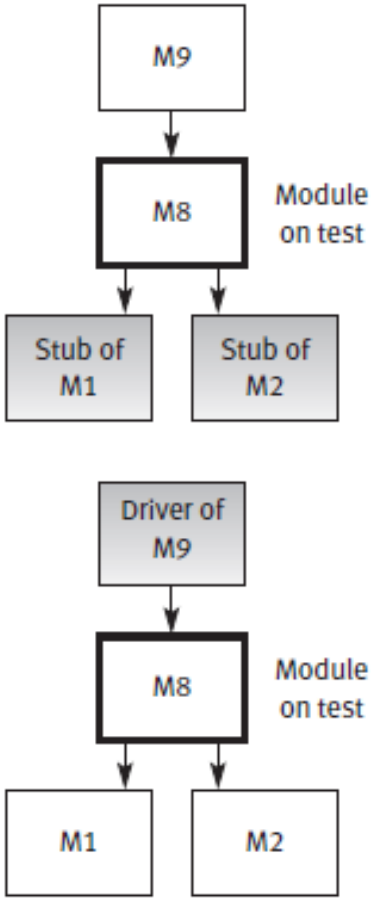
- Stubs are dummy modules
- Replacing unavailable lower level modules
- Required for top-down testing of incomplete systems
- Stub provides the results of subordinate module, yet to be coded.



From Galin, 2004

Incremental: Drivers

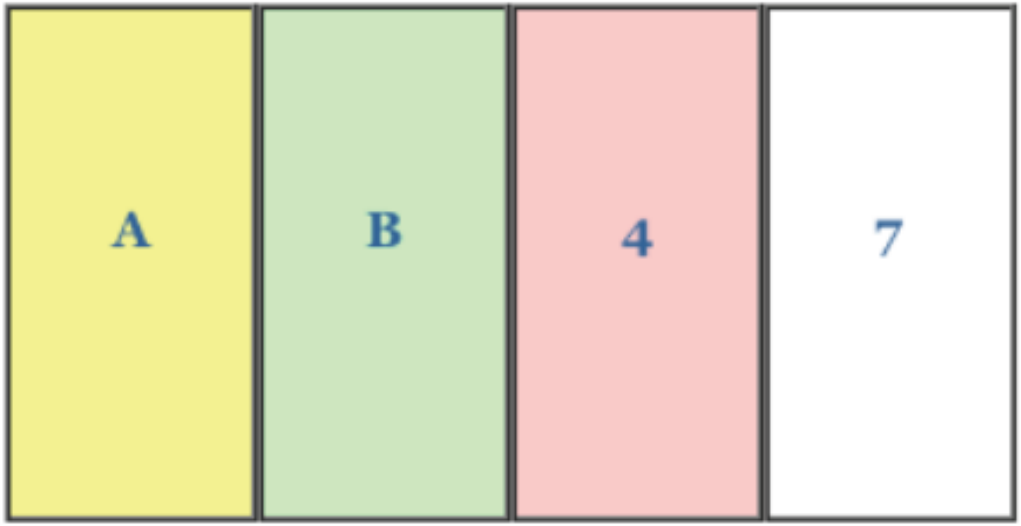
- Required in bottom-up testing
- Substitute module, but of the upper-level module



From Galin, 2004

Big bang vs. Incremental

- Big bang
  - Effective in small and simple programs
  - Prospects of keeping on schedule and within budget are substantially reduced.
  - Despite vast resources invested, effectiveness of the approach is insufficient.
- Incremental
  - Usually on small software modules, as unit or integration tests.
  - Easier to identify higher % errors. → prevents mitigation of errors to later stages in a lifecycle
  - Identification and correction of errors are much simpler with fewer resources (limited volume of software).
  - Increased quantity of programming resources for preparation of stubs/drivers.
  - Increased number of testing operations for a single program



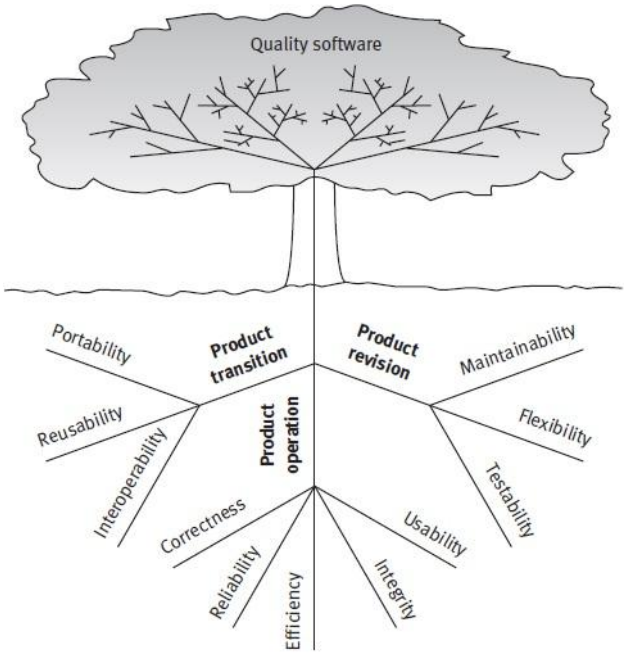
"If a card has a vowel on one side, then it has an even number on the other side."

Which card(s) must you turn over to determine the following statement is FALSE?

a) A and 4?
b) B and 4?
c) A and 7?
d) B and 7?
e) A and B?
f) 4 and 7?

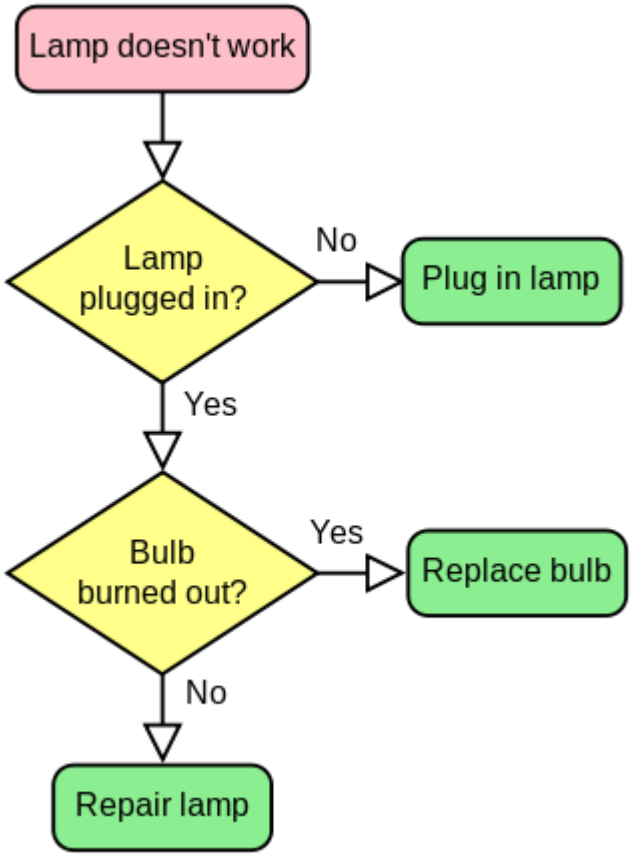
Software test classifications

- According to testing concept
  - Black-box → erroneous outputs
  - White-box → glass-box
- According to requirements
  - Based on McCall's quality model
    - For "correctness"
    - Output correctness tests
      - Documentation tests
      - Availability tests
      - Data processing and calculations correctness tests



Flow charts to represent basic testing path

- Diamonds: Options covered by conditional statements
- Rectangles: Software sections connecting those conditional statements

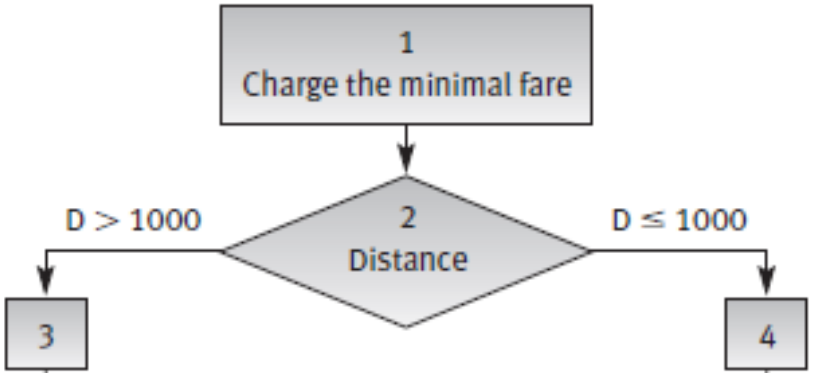


Example: Imperial Taxi Services (ITS) taximeter

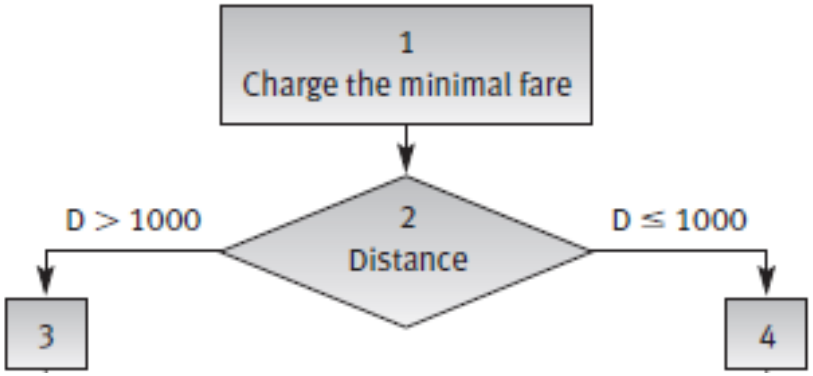
- Serves one-time passengers and regular clients.

1) Minimal fare: $2. This fare covers the distance traveled up to 1000 yards and waiting time (stopping for traffic lights or traffic jams, etc.) of up to 3 minutes.
2) For every additional 250 yards or part of it: 25 cents.
3) For every additional 2 minutes of stopping or waiting or part thereof: 20 cents.
4) One suitcase: no charge; each additional suitcase: $1.
5) Night supplement: 25%, effective for journeys between 21.00 and 06.00.
6) Regular clients are entitled to a 10% discount and are not charged the night supplement.

1) Minimal fare: $2. This fare covers the distance traveled up to 1000 yards and waiting time (stopping for traffic lights or traffic jams, etc.) of up to 3 minutes.
2) For every additional 250 yards or part of it: 25 cents.

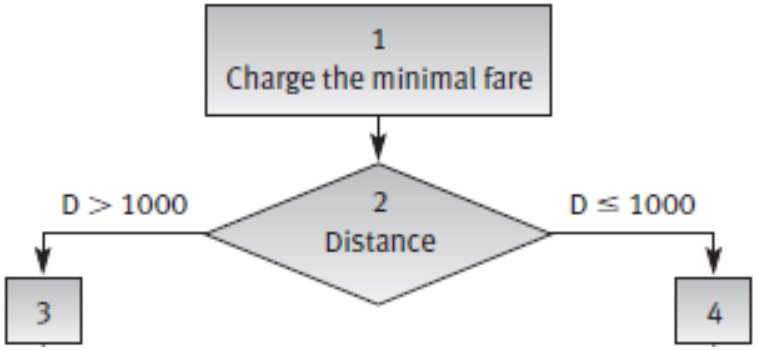


1) Minimal fare: $2. This fare covers the distance traveled up to 1000 yards and waiting time (stopping for traffic lights or traffic jams, etc.) of up to 3 minutes.
2) For every additional 250 yards or part of it: 25 cents.



Quiz: Complete the flow graph

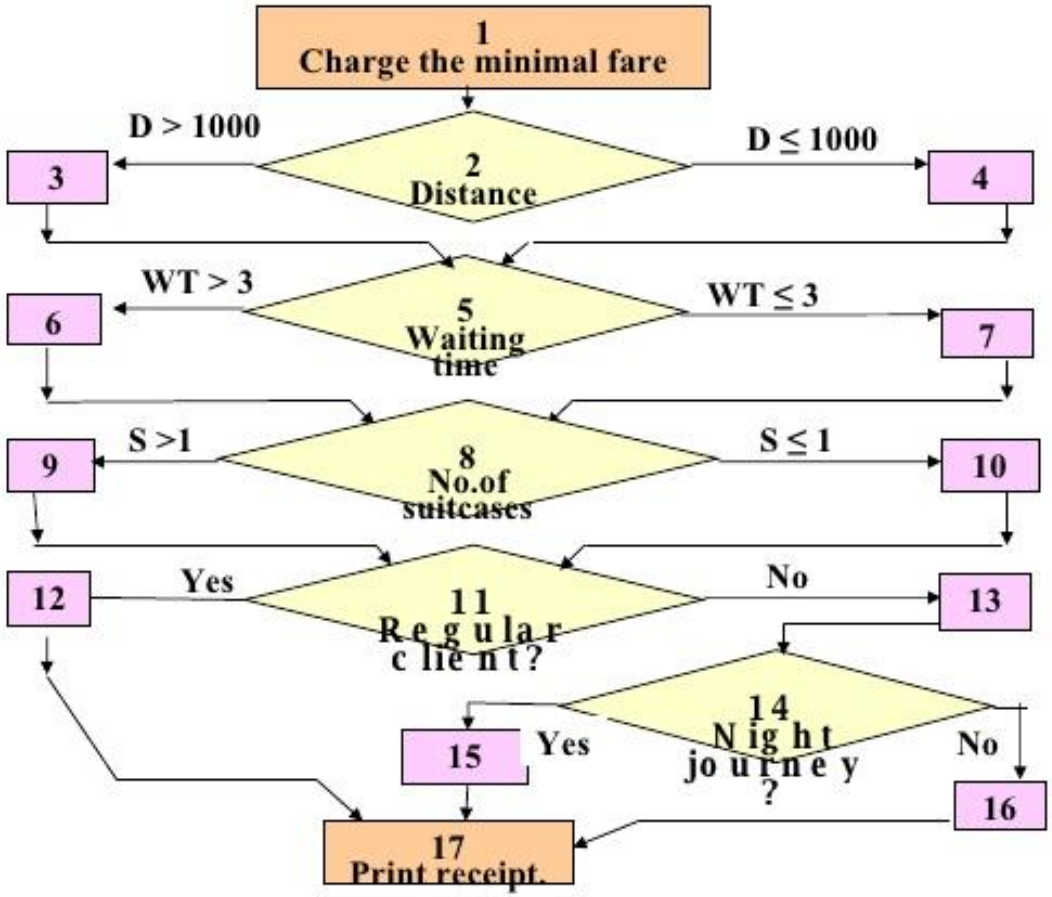
QUIZ: Complete the testing flow graph



1) Minimal fare: $2. This fare covers the distance traveled up to 1000 yards and waiting time (stopping for traffic lights or traffic jams, etc.) of up to 3 minutes.
2) For every additional 250 yards or part of it: 25 cents.
3) For every additional 2 minutes of stopping or waiting or part thereof: 20 cents.
4) One suitcase: no charge; each additional suitcase: $1.
5) Night supplement: 25%, effective for journeys between 21.00 and 06.00.
6) Regular clients are entitled to a 10% discount and are not charged the night supplement.

OHT 9.11

ITS - Flow chart



Galin, *SQA from theory to implementation*

White box testing (structure-based)

+ Statement-by-statement checking of code
+ Performance of line coverage follow-up
  - LOC that have not yet been executed
+ Quality of coding
- Vast resources utilized (above those in black-box)
- Inability to test software performance in terms of availability, reliability, other testing factors.

Black-box testing

- Tests other than output correctness and maintainability tests are all considered as *black-box*.

+ Fewer resources

- Coincidental aggregation of several errors may produce the correct response for a test case.

Equivalence class partitioning (black-box)

Output correctness: Greater consumption of test resources

- Equivalence class (EC)
  - Set of input variable values that produce the same output results or that are processed identically.
- Valid EC: only valid states
- Invalid EC: only invalid states

Rules:

- Test cases are defined for *valid ECs* and *invalid ECs.*
- Test cases are added as long as there are uncovered ECs.
- Total number of test cases for valid ECs <= valid ECs
- One test case for each invalid EC.

Example 1

- Program that is supposed to accept any number between 1 and 99
- How many equivalence classes?

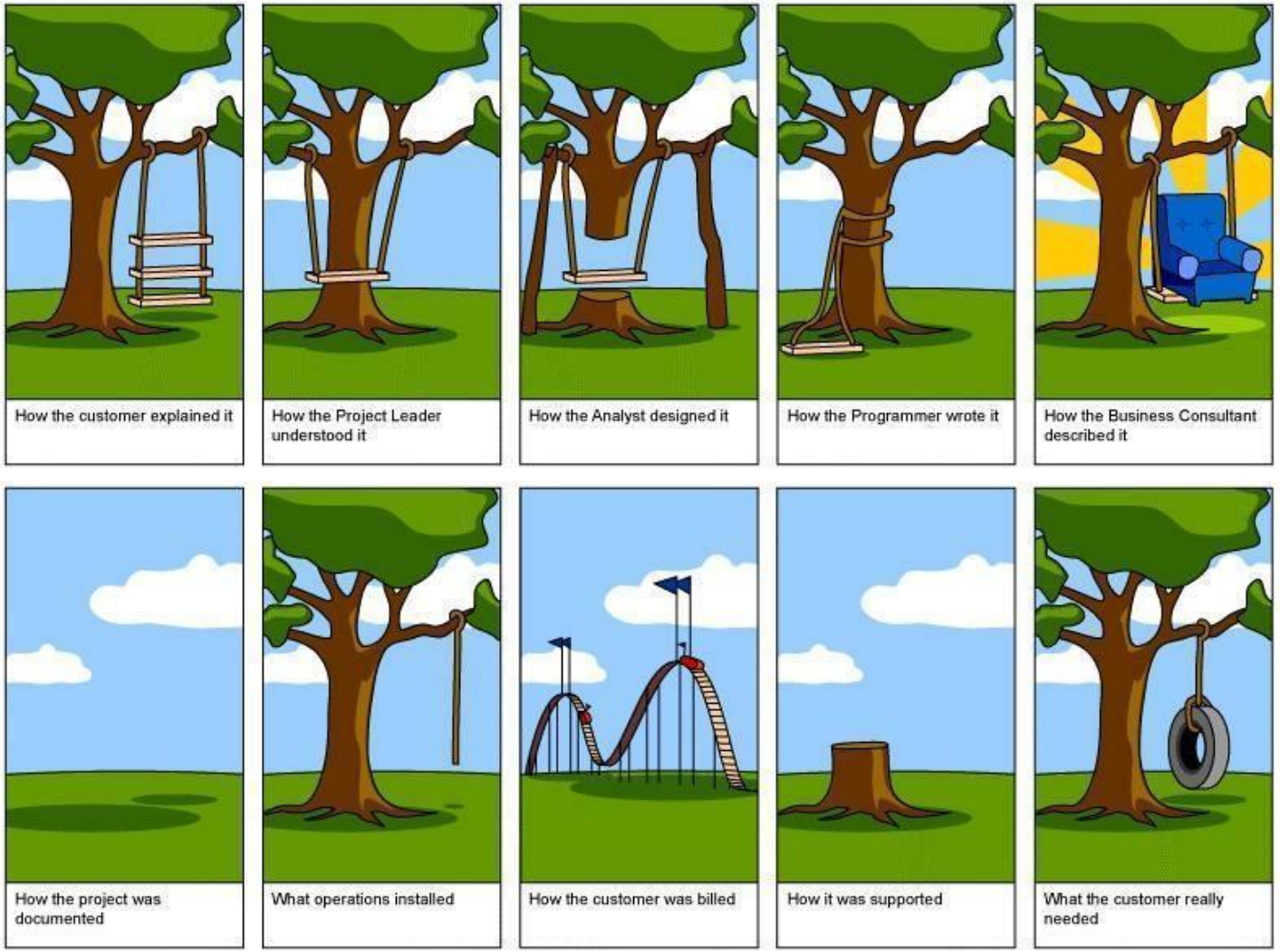
Example 1 (cont'd)

- Program that is supposed to accept any number between 1 and 99
- How many equivalence classes?
- 4 EC
  - Valid: Any number between 1 and 99
  - Invalid: Not a number
  - Invalid: Any number less than 1
  - Invalid: Any number greater than 99

Equivalence classes

- Additional hints
  - (1) Look for *extreme* range of variables
    - Try very small numbers to check if sign works
    - Try very large 16-bit and 32-bit integers
  - (2) Look for *maximum size of memory* variables
    - e.g. for a depth-first search, set the length of the path to be very long and check for memory overflow.
  - (3) One EC must include *all members of a group*
    - e.g. for a program accepting usernames as input
      - All string of alphabet (starting with capital or lower case letters)
      - Limit or extend the size of the character
      - Check for non-ascii characters

Test Levels


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Types of testing

- Unit test
- Integration test
- System test
- Acceptance (α,β) test
- Regression test
- Others (load, etc.)

- 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 types

- Unit
  - Internal behavior of individual units against specifications (functionality)
- Integration
  - Test interactions between modules based on design/interface specifications (stubs and drivers)
- System
  - Verification of the completed system functionality as a whole against requirement specifications.

Who performs the tests?

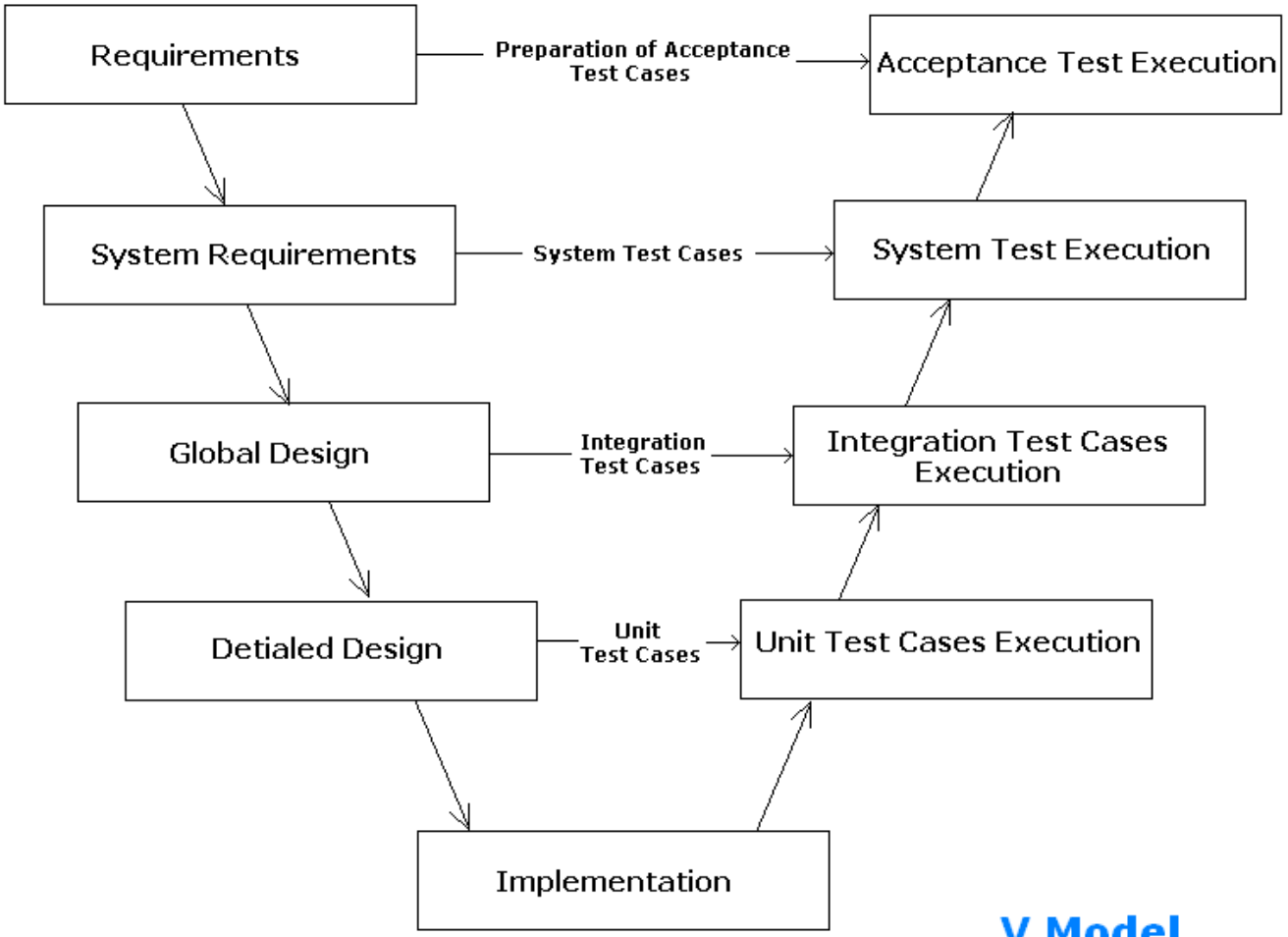
- Unit and integration tests
  - Software development team (developer task)
  - Often, integration test team
- System tests
  - Independent testing team (internal or external)
  - Vendor
- Acceptance
  - Customer

Acceptance tests

- Validation of complete system against actual needs
  - Functional
  - Non-functional
  - Black-box
- α-test: environment controlled by vendor
- β-test: real settings of customer

Regression tests

- Recheck all test cases passed previously
- Corrections have been performed satisfactorily and have not intentionally introduced new errors
- Library of test suites
  - Automated tools
  - Subset of test suites


Requirements
Preparation of Acceptance Test Cases
Acceptance Test Execution
System Requirements
System Test Cases
System Test Execution
Global Design
Integration Test Cases
Integration Test Cases Execution
Detialed Design
Unit Test Cases
Unit Test Cases Execution
Implementation

V Model

References

- Galin, D. 2004. Software Quality Assurance: From theory to implementation, Addison Wesley.
- Turhan, B. 2011. Software Quality and Testing course, Lecture Notes 7.
- Bener, A. 2006. Software testing: Test case design, Lecture Notes.
- Far, B.H. Software Reliability & Software Quality, Chapter 11: Preparing and Executing Test, Dept. of Electrical and Computer Engineering, University of Calgary.