COMP 3711

(OOA and OOD)

Software Testing 2
Test Model

Testing Computer Software (2ed) Kaner/Falk/Nguyen (Chapters 12, 7)

Need for a Model

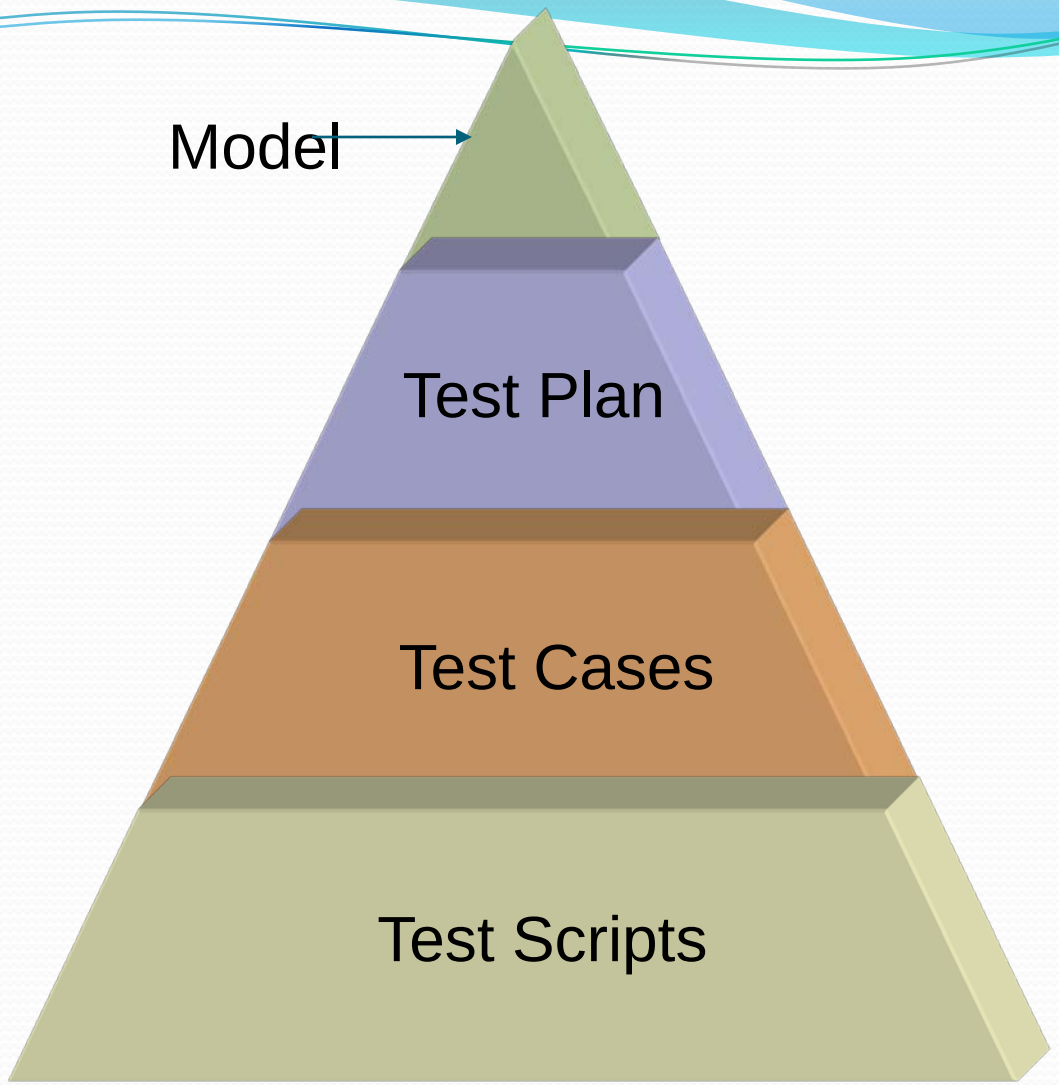
- Testing is a complex and confusing process
- Huge domain of possible tests that can be done
- Object of model is to illuminate the relative importance of various tasks that could be performed at some particular time in the development cycle

Start With Programmers

- Find as many problems as early in the process as possible
  - major motherhood issue
- Starts with programmers
  - Without programmers dedicated to the production of a quality product, testing isn't going to work
  - Requires management commitment as well
  - Programmers should not depend on testers to find bugs

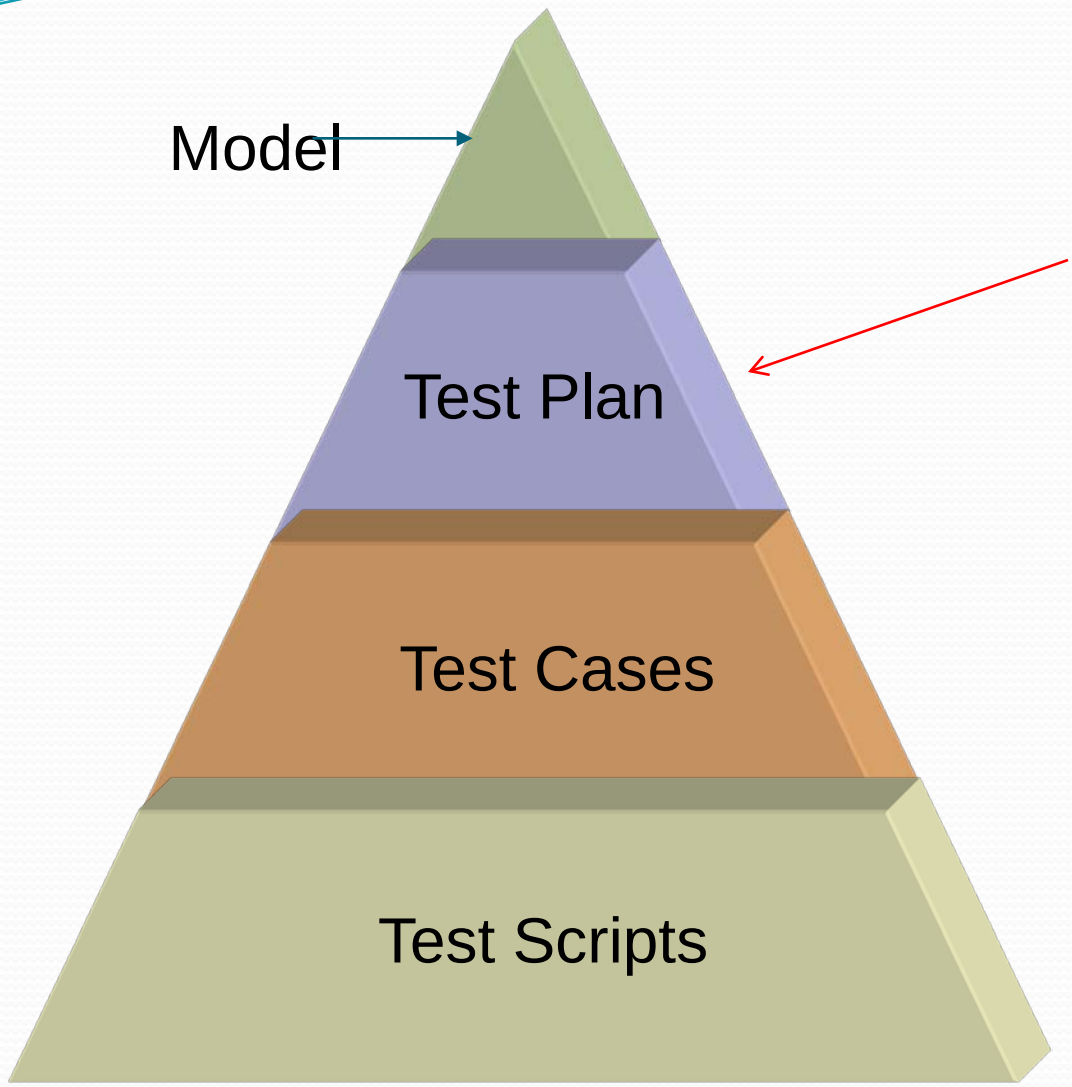
Early Programmer Intervention

- Both managers and testers should encourage programmers to produce quality code
- Quality of output must be at least as important as volume of output
- Tool support is required
- Things like Quantify and Purify
- Make quality code part of the culture


Model
Test Plan
Test Cases
Test Scripts

Test Strategy – Refining the Model

- Statement of the overall approach to testing,
- Identifying what:
  - Levels of testing are to be applied
  - Methods, techniques and tools to be used.
- A test strategy should ideally be organization wide
  - Applicable to all of organizations' software development.
- Critical to the success of software development within the organization.



What is a Test Plan?

Test Plan: Document

- Development of a test plan should start in concert with the system design.
  - Facilitates the technical tasks of testing
  - Improves communication about testing tasks and products
  - Provides structure for organizing, scheduling and managing the test project

Test Plan: Document

- Describes:
  - scope, approach, resources, schedule of intended testing activities
- Identifies:
  - test items, features to be tested, testing tasks and sequence, who will do each task, any risks requiring contingency planning, description of the test environment

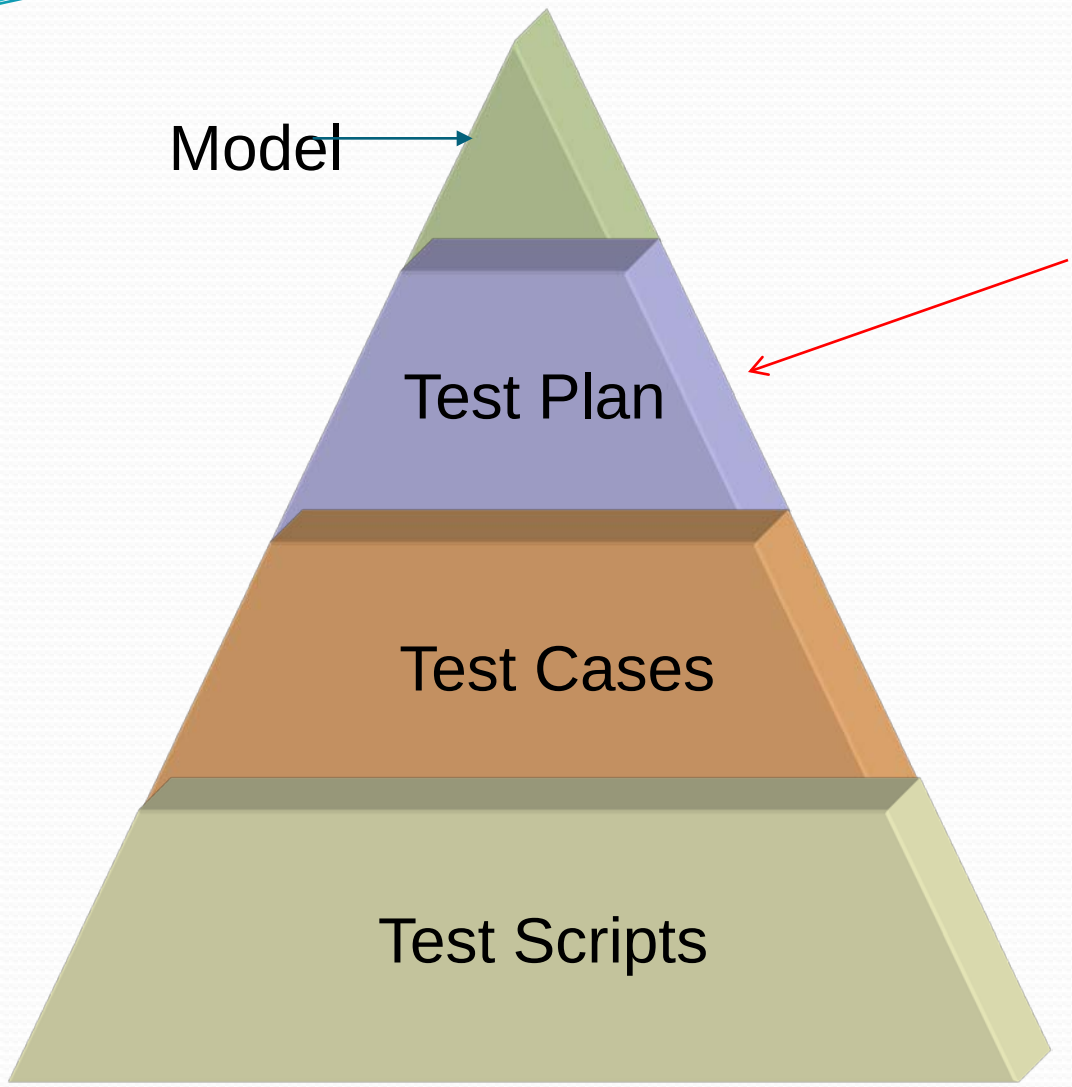
Ideal Test Plans

- Traditionally huge, impressive detailed test planning documents were expected

- ANSI/IEEE 829 requires
  - Test design specs
  - Test case specs
  - Test logs
  - Test-various-identifiers
  - Test procedure specs
  - Test item transmittal reports
  - Input/output specs
  - Special procedure requirements specs
  - Intercase dependency notes
  - Test deliverable lists
  - Staff plans

Realistic Test Plans

- A valuable tool that helps:
  - Manage the testing project
  - Find bugs in the product
- Facilitates the technical tasks of testing
- Improves communication about testing tasks and products
- Provides structure for organizing, scheduling and managing the test project



What can a test plan do for your software development project?

Facilitate the Technical Tasks of Testing

- A Test Plan Improves testing coverage
  - Build a giant list of all features, reports, error messages, supported printers, menu choices, dialog boxes, options
  - Keep the list up-to-date
  - Make sure that they're all tested
- Avoid unnecessary repetition
- Analyze program and spot good test cases

Continue ….

- Provide structure for the final test
  - Help decide which of all possible tests must be run before release
- Improve test efficiency
  - Idea is to reduce the number of tests run without reducing the number of bugs found
  - Identify test cases that should produce the same result
  - Just run one of them

Continue ….

- Check for Completeness
  - Overlooked area of program
  - Overlooked Class of Bugs
    - Best to build a giant potential bug list
  - Overlooked class of test
    - Volume or load test, test with background printing
  - Simple oversight

Improve Communication About Testing

- Get feedback from readers about accuracy and coverage of testing
  - Assuming that anybody reads the test plan
- Communicate size of testing job
- Get feedback about testing depth and timing
- Modularize and divide up the testing work

Provide Structure for Organizing, Scheduling and Managing

- Reach agreement about testing tasks
  - Specifies what will and what won't be tested
- Identify testing tasks
- Identify relationships between tasks
- Organize and coordinate
  - Decide who will do each test and what resources will be required

Continue ….

- Improve individual accountability
  - When horrible bugs start crawling out of the woodwork, it's possible to assess whether or not the test plan:
    - would have found them
    - should have found them
    - just plain missed them

Continue ….

- Even conscientious testers have been known to skip tests if:
  - tests are unnecessarily redundant or boring
  - A lot of forced unpaid overtime is being worked

Continue ….

- Measure project status and improve project accountability
  - Based on the test plan, it's possible to see:
    - What has really been done
    - Depending on the test plan, what remains to be done
    - Can check progress on testing against schedule

Evolutionary Development of a Test Plan

- Testers may not have a complete specification for the product when they must start planning the testing
- Work on test plan and actual testing at the same time
- Write a little bit of the test plan
- Try it out
- Modify test plan based on results

Getting Started

- Go through current program at superficial level
  - Try to maintain uniform superficial level
  - Test against the docs, assuming any exist
  - Build a list of the program's functions and test them
  - Investigate limits/boundary conditions

- Build a foundation
  - Test the whole program , but not very thoroughly

Where to Focus

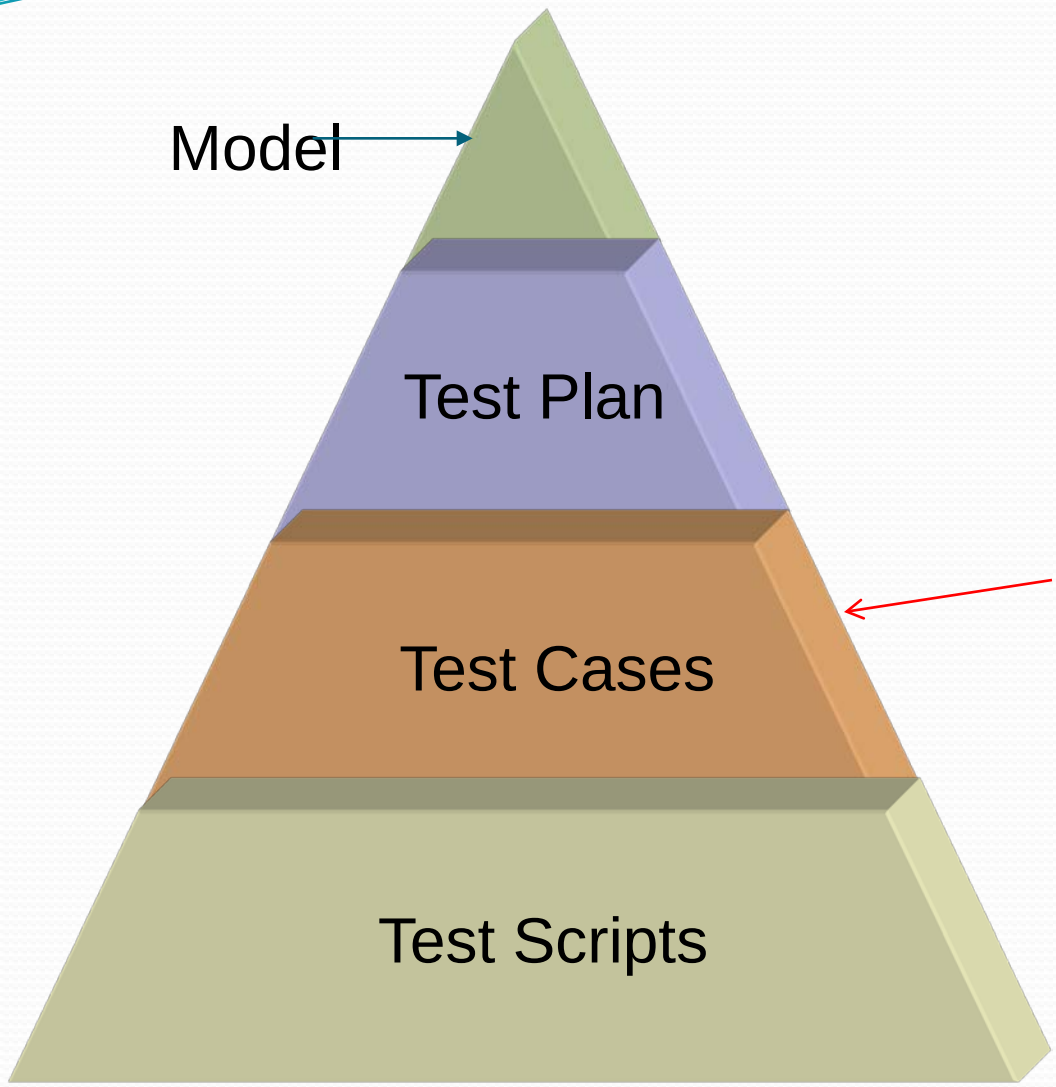
- Most likely errors
  - Go where the bugs are
  - 47% of errors found in 4% of modules
- Most visible errors
  - Which errors will customers find first
  - Areas of critical functionality
- Most often used program area

Where to Focus … conti..

- Distinguishing area of program
  - Make sure that the claim to fame actually works as advertised
  - Heavily optimized code areas are often hard to fix
- Hardest areas to fix
  - Ask the programmers
  - Find bugs here early, they'll be glad you did
- Area best understood by tester
  - Can test effectively here while learning rest of program

Detailed Components of Test Plans

- Show only what you need to know to test the program
- Can be built from a specification
- Building these components is a very useful way to review the spec and find the holes
- More often built as the program evolves
- Most of the information comes from running the program



What is a test case?

Test Case

- A commonly used term for a specific test
- This is usually the ***smallest unit of testing***
- Consists of information such as:
  - Requirements tested, test steps, verification steps, prerequisites, outputs, test environment state
  - Set of inputs, execution preconditions, and expected outcomes
- Developed for a particular objective:
  - e.g. exercise a particular program path
  - e.g. verify compliance with a specific requirement

Test Case Development

- Test the things that are most important first.
  - Often used ***scenarios***
  - ***Equivalence classes*** and ***components*** involved in these
- Can test the scenario with different equivalent classes (one test case for each).
- Could include several equivalent classes in a scenario but make sure your test case doesn't become too complicated.

Test Case Development

- Test Case: series of one or more scripts that is run as one test
- Covers one scenario or a series of equivalence classes or one component
  - Scenarios first
- Can mean different things
  - From starting up program? Or assume program started?
  - Very long, very short
- Should be easily distinguishable

Test Case Development: *Scenario*

- A series of steps that produce an often used result
- Hotel front desk example:
  - Customer enters with reservation for tonight
  - Script Ad Hoc
    - Lookup customer name (reservation number)
    - Find room assigned to customer
    - Change room at customers request
    - Mark customer and room as “IN”
    - Print key for customer
    - Mark the customer as In
  - From here can create detailed script

Test Case Development: ***Series of Equivalence classes***

- Example: For a program that accepts numbers from 1 to 99 as input, there are 4 equivalence classes
  - Correct input
  - Numbers less than 1
  - Numbers greater than 99
  - Inputs that are not numbers
- Create detailed Script (s) from:
  - enter number < 1
  - enter number > 99
  - enter non-number
  - Enter valid input

Test Case Development: *One Component*

- Exercise a component of the system
- Example is the List Component in the Scribble example.

Scribble

- We can add an item, delete an item , move an item and change the order of the list

A Good Test Case

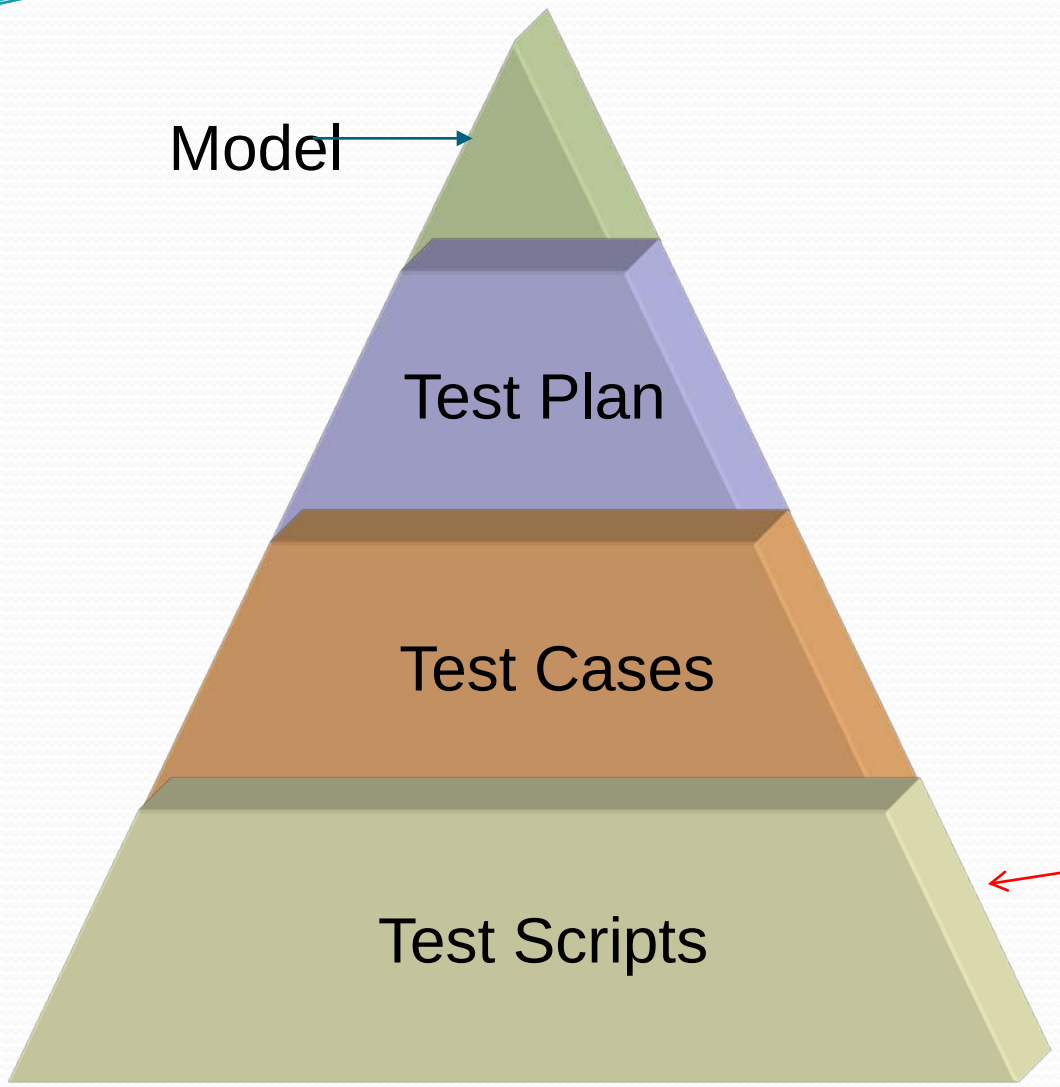
- Has a reasonable probability of finding errors
  - Can work backwards from suspected failures
- Not redundant
  - If 2 tests check for the same error, why run both
- Best in its class
  - Want test that's most likely to find error
  - Boundary values better than non-boundary

Continue …

- Neither too simple nor too complex
  - Don't try to save time by combining too many test cases initially
- Makes program failures obvious
  - Write down expected output or result when test is created
  - Make output as short as possible

Matrices

- Some examples:
  - *Test Matrix for Input Field (e.g. for unit test)*
  - *Test Matrix for Repeating issues (e.g. for unit test, system test)*
  - *Traceability Matrix for specification-based testing showing relationship between test requirements and test cases (e.g. for functional testing, system test)*
  - *Error Catalogs (e.g. for test plan)*



What is a test script?