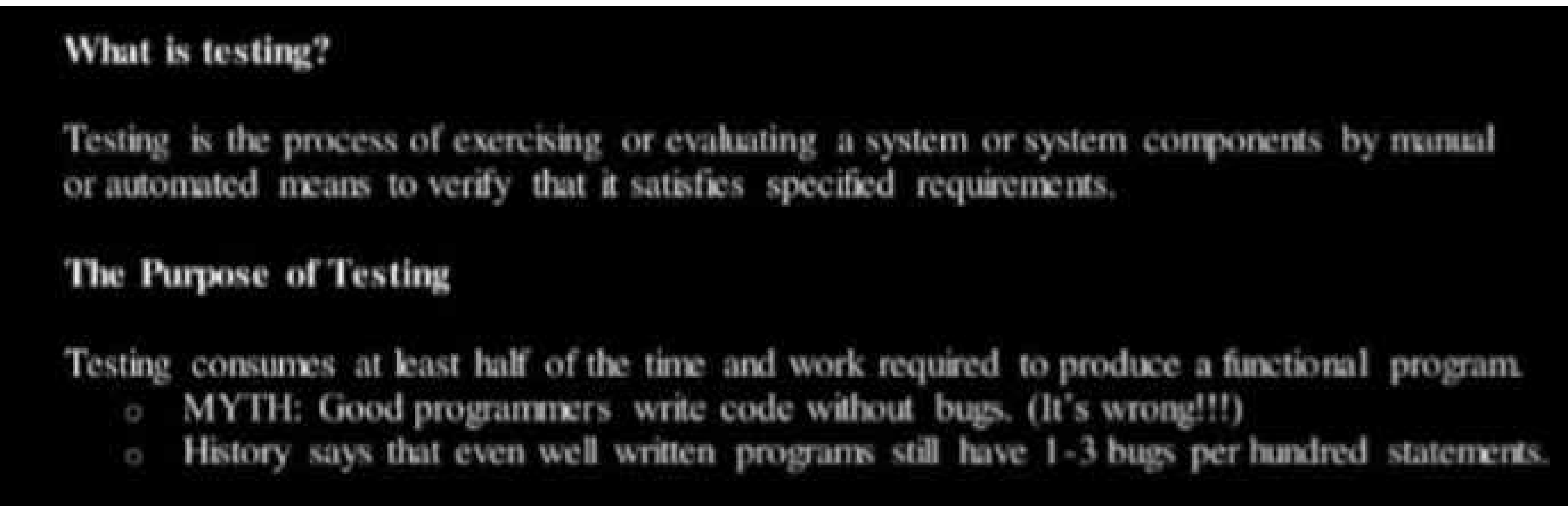

What is testing?

Testing is the process of exercising or evaluating a system or system components by manual or automated means to verify that it satisfies specified requirements.

The Purpose of Testing

Testing consumes at least half of the time and work required to produce a functional program.

- MYTH: Good programmers write code without bugs. (It's wrong!!!)
- History says that even well written programs still have 1-3 bugs per hundred statements.

CONSEQUENCES OF BUGS:

- **Importance of bugs:** The importance of bugs depends on frequency, correction cost, installation cost, and consequences.
    1. **Frequency:** How often does that kind of bug occur? Pay more attention to the more frequent bug types.
    2. **Correction Cost:** What does it cost to correct the bug after it is found? The cost is the sum of 2 factors: (1) the cost of discovery (2) the cost of correction. These costs go up dramatically later in the development cycle when the bug is discovered. Correction cost also depends on system size.
    3. **Installation Cost:** Installation cost depends on the number of installations: small for a single user program but more for distributed systems. Fixing one bug and distributing the fix could exceed the entire system's development cost.
    4. **Consequences:** What are the consequences of the bug? Bug consequences can range from mild to catastrophic.

  A reasonable metric for bug importance is
  Importance= ($) = Frequency * (Correction cost + Installation cost + Consequential cost)

- **Consequences of bugs:** The consequences of a bug can be measure in terms of human rather than machine. Some consequences of a bug on a scale of one to ten are:
    1 **Mild:** The symptoms of the bug offend us aesthetically (gently); a misspelled output or a misaligned printout.
    2 **Moderate:** Outputs are misleading or redundant. The bug impacts the system's performance.
    3 **Annoying:** The system's behavior because of the bug is dehumanizing. *E.g.* Names are truncated or arbitrarily modified.
    4 **Disturbing:** It refuses to handle legitimate (authorized / legal) transactions. The ATM won't give you money. My credit card is declared invalid.
    5 **Serious:** It loses track of its transactions. Not just the transaction itself but the fact that the transaction occurred. Accountability is lost.
    6 **Very Serious:** The bug causes the system to do the wrong transactions. Instead of losing your paycheck, the system credits it to another account or converts deposits to withdrawals.
    7 **Extreme:** The problems aren't limited to a few users or to few transaction types. They are frequent and arbitrary instead of sporadic infrequent) or for unusual cases.
    8 **Intolerable:** Long term unrecoverable corruption of the database occurs and the corruption is not easily discovered. Serious consideration is given to shutting the system down.
    9 **Catastrophic:** The decision to shut down is taken out of our hands because the system fails.
    10 **Infectious:** What can be worse than a failed system? One that corrupt other systems even though it does not fall in itself ; that erodes the social physical environment; that melts nuclear reactors and starts war.

BASICS OF DATA FLOW TESTING:

DATA FLOW TESTING:

- Data flow testing is the name given to a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of data objects.
- For example, pick enough paths to assure that every data object has been initialized prior to use or that all defined objects have been used for something.
- **Motivation:** It is our belief that, just as one would not feel confident about a program without executing every statement in it as part of some test, one should

not feel confident about a program without having seen the effect of using the value produced by each and every computation.

DATA FLOW MACHINES:

- There are two types of data flow machines with different architectures. (1) Von Neumann machines (2) Multi-instruction, multi-data machines (MIMD).
- **Von Neumann Machine Architecture:**
  - Most computers today are von-neumann machines.
  - This architecture features interchangeable storage of instructions and data in the same memory units.
  - The Von Neumann machine Architecture executes one instruction at a time in the following, micro instruction sequence:
    - Fetch instruction from memory
    - Interpret instruction
    - Fetch operands
    - Process or Execute
    - Store result
    - Increment program counter
    - GOTO 1
- **Multi-instruction, Multi-data machines (MIMD) Architecture:**
  - These machines can fetch several instructions and objects in parallel.
  - They can also do arithmetic and logical operations simultaneously on different data objects.
  - The decision of how to sequence them depends on the compiler.

UNIT III
DOMAIN TESTING

DOMAINS AND PATHS:

- **INTRODUCTION:**
  - **Domain:** In mathematics, domain is a set of possible values of an independent variable or the variables of a function.
  - Programs as input data classifiers: domain testing attempts to determine whether the classification is or is not correct.
  - Domain testing can be based on specifications or equivalent implementation information.
  - If domain testing is based on specifications, it is a functional test technique.
  - If domain testing is based implementation details, it is a structural test technique. o For example, you're doing domain testing when you check extreme values of an input variable.

  All inputs to a program can be considered as if they are numbers. For example, a character string can be treated as a number by concatenating bits and looking at them as if they were a binary integer. This is the view in domain testing, which is why this strategy has a mathematical flavor.

- **THE MODEL:** The following figure is a schematic representation of domain testing.

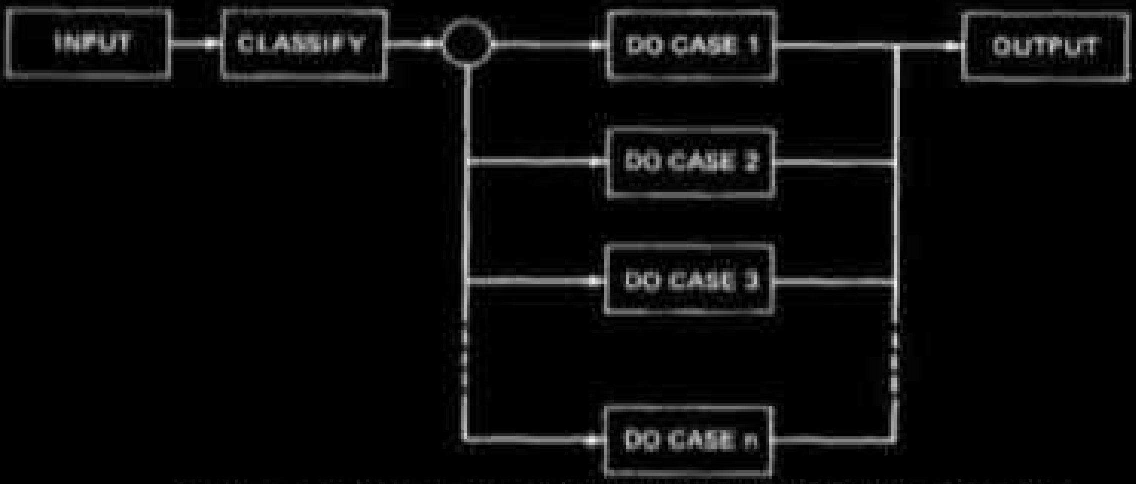

Figure 4.1: Schematic Representation of Domain Testing.

  - Before doing whatever it does, a routine must classify the input and set it moving on the right path.
  - An invalid input (e.g., value too big) is just a special processing case called 'reject'.
  - The input then passes to a hypothetical subroutine rather than on calculations.
  - In domain testing, we focus on the classification aspect of the routine rather than on the calculations.
  - Structural knowledge is not needed for this model - only a consistent, complete specification of input values for each case.
  - We can infer that for each case there must be at least one path to process that case.