Software validation and Testing

Lecture-1

Belaynesh C.

Software

- A software is simply a collection of programs.

Example , MS- Office, consisting of Ms-Word, Ms -Excel etc.

- A program is set of instructions that carry out a certain task with the given input data.

For example, to open file, find a specific string and replace it with another string.

- Results obtained from a program execution are expected to satisfy a developer (programmer) and general user.

- Testing is a **critical element** of software development life cycle where a program is executed by a programmer or professional tester to at least locate possible errors, failure and fault. However, higher level objectives of testing are risk reduction and quality software production.

- In general, there is no guarantee that our programs are error free. It is always important to aim to discover the maximum number of defects than be content with the fact that a software is working as required.

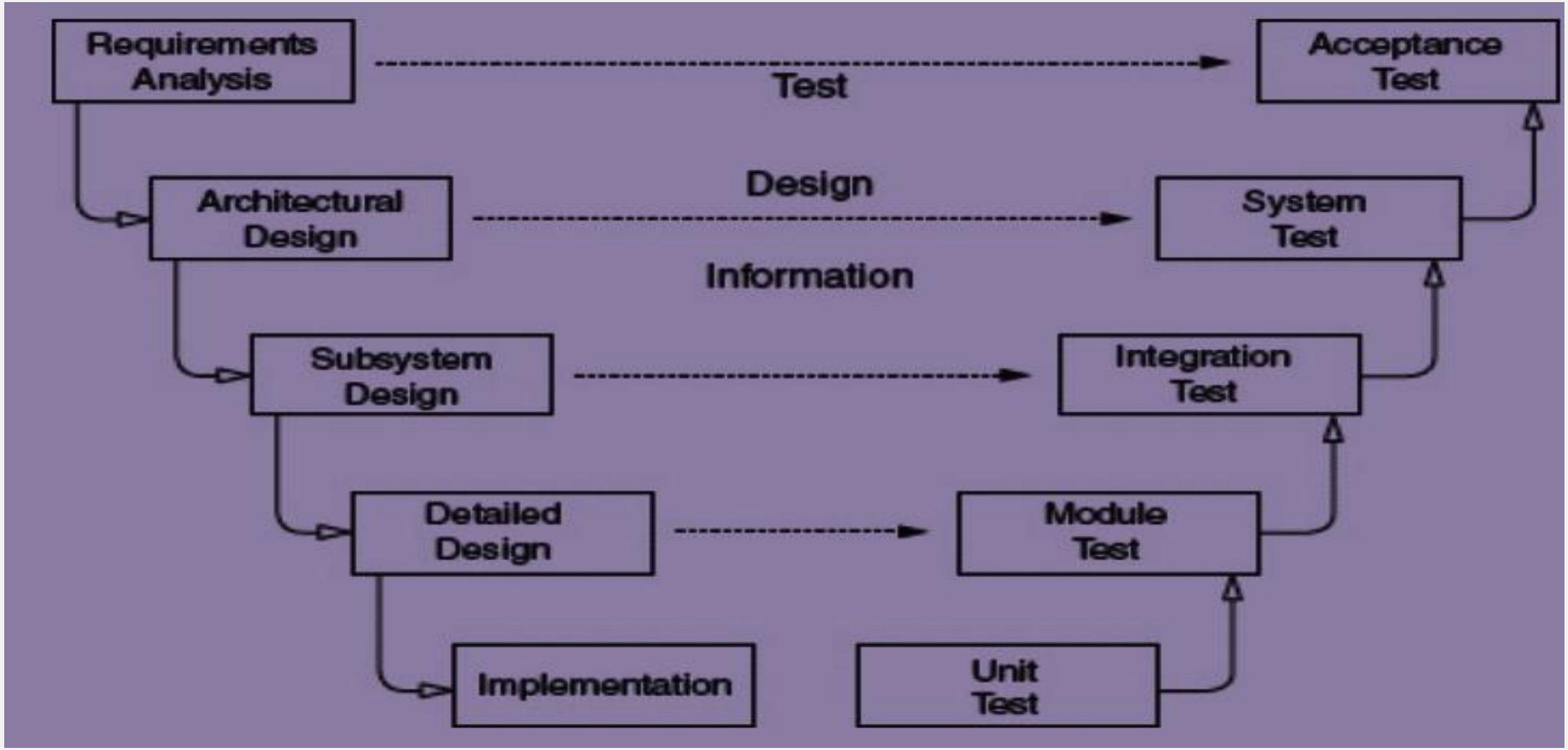
Level0: There is no difference between testing and debugging.
Level1: The purpose of testing is to show correctness.
Level2: The purpose of testing is to show that the software does not work.
Level3: The purpose of testing is not to prove anything specific, but to reduce the risk of using the software.
Level4: Testing is a mental discipline that helps all IT professionals develop higher-quality software.

- Sometimes called **software quality control** or **software quality assurance.**

  →Basic goals: **validation** and **verification**
  validation: **are we building the right product?**
  verification: **does "X" meet its specification?**
  where "X" can be code, a model, a design diagram, a requirement.

  →At each stage, we need to verify that the thing we produce accurately represents its specification.

Software Development and testing activities: the “V” model



Error: internal incorrect state
Fault: static defect
Failure : external incorrect behavior

Example: Let *S* be the space defined by a variable of type integer, and let *R* be the following specification:

$$R = \left\{ (s, s') \mid s' = (s+1)^2 \bmod 3 + 12 \right\}$$

and let *p* be the following program on space *S*:

```
{
s=s+1;
s=2*s
s=s%3
s=s+12;
}
```

- For initial state s=1, line 1 produces state s=2, which line 2 maps into 4. Because for s=2, 2*s and s*s are the same, the fault has no impact on this execution.

- We say that the *fault is not sensitized*, that is, it is not causing the state to be different from the correct state.

- For initial state s=2, line 1 produces state s=3, which line 2 maps onto 6. Because the correct state at this step of the computation is 3*3=9 rather than 2*3=6, we say that we are observing an *error*.

- In this case, we say that the *fault has been sensitized*: unlike in the previous case, it has produced a state that is distinct from the expected state at this step.

- To identify a fault individual statements of program parts are judged, the identification of an error refers to a sequence of expected/correct states against which actual states are compared.

- Even though the fault is sensitized and generates an error, it causes no long-term impact, since by line 3 the program generates a correct state: indeed (6 mod 3) is the same as (9 mod 3).
- So at the end of the program (after line 4), the state of the program is s=12. We say that the *error has been masked*.

• For initial state s=3, line 1 produces state s=4, which line 2 maps onto state s=8. Because the correct state at this step of the computation ought to be s=16 instead of s=8, we observe again that the fault has been sensitized and has caused an error.

By line 3, the state becomes (8 mod 3)=2 rather than (16 mod 3)=1, hence the *error is propagated* to the next state.

By line 4, the state becomes s=2+12=4 rather than s=1+12=13. The error is again propagated causing a *failure* of the program.

- The *fault* is the statement in line 2, which should be (s=s*s) rather than (s=2*s);

- An *error* is the impact of the fault on program states; not all executions of the program give rise to an error; those that do are said to sensitize the fault. Executions on initial states s=2 and s=3 lead to errors.

- A failure is the event whereby an execution of the program on some initial state violates the specification; not all errors give rise to a failure; those that do are said to be propagated; those that do not are said to be masked.

Failure risks

- Inconvenience: Some low-level errors cause discomfort to end users.
- Financial Loss: big businesses are prone to huge financial losses incurred as a result of insufficient testing.
- Life Loss: Faulty instructions on cars, airplanes, medical appliances

Evolution of software testing

- Program execution on a machine: before 1970
- Operating system based: not enough for application specific software
- Manual testing
- Automated testing
- Continuous testing
- Artificial Intelligence

Is software testing becoming easier?

- Currently, almost every aspect of our life is involved with software directly or indirectly. Proportionally, development tools and frameworks are available which have been tested previously. Such opportunities reduce testing complexity.

Principles of software testing

1. Testing identifies the presence of defects
Testing enables the identification of defects present in a piece of code, but does not show that such defects are not present.

2. Exhaustive testing is impossible
It is impossible to undertake exhaustive testing, to test “everything”.

- When testing a small calculator, “testing everything” would mean trying all combinations of input values, actions, configurations (hardware and software), resulting in an almost infinite number of test cases, which is not realistic to design or to execute.

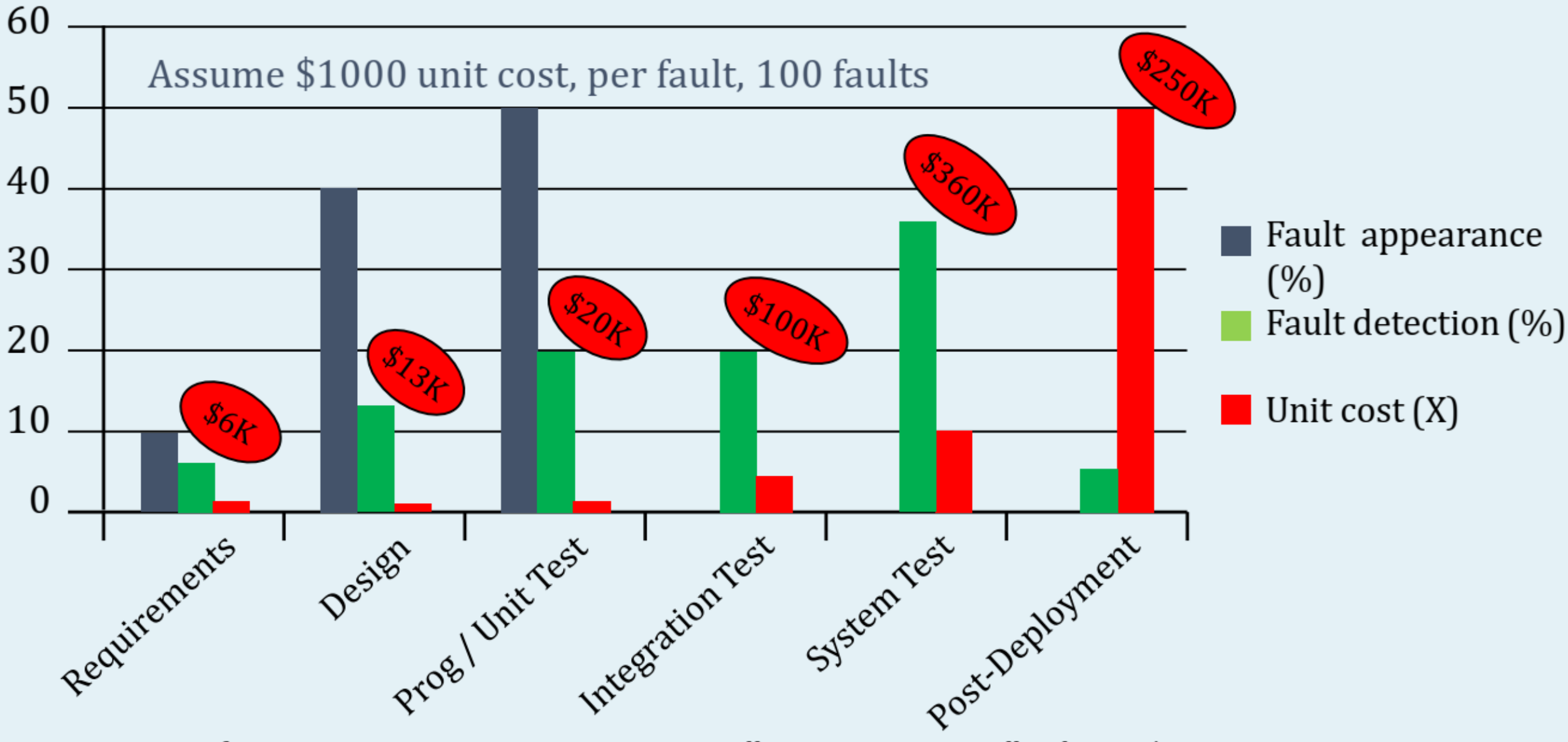
- Exhaustive testing is not possible, except in some rare trivial cases.

3. Early testing

Test early in the development cycle, or “early testing”. This principle is based on the fact that costs, whether development costs, testing costs, defect correction costs, etc. increase throughout the project.

It is economically more sensible to detect defects as early as possible; thus avoiding the design of incorrect code, the creation of test cases, which require design and maintenance of the created test cases, the identification of defects that have to be logged, fixed, and retested.

Cost of Late testing



4.Defect clustering
This principle means also that if you find a defect, it might be efficient to look for other defects in the same piece of code.

5. Pesticide paradox
Fifth principle: the pesticide paradox, or lack of efficiency of tests when they are used over a period of time.

It is strongly suggested that tests are changed over time, so as to modify them and so that the impact of this principle will be negligible. Uses of other test techniques, variation in test data, or in the order of execution of the tests are ways to counter this principle.

Reuse, without modification, of tests can be necessary to ensure that no side effects (i.e. regression) have been introduced in the software as result of changes in that software or in its execution context.

6. Testing is context dependent

Safety-critical systems will be tested differently and with a more intense effort, than most e-commerce systems. However, this principle goes much further, because it also applies to the evolution of the context of your own software application.

7. Absence of errors fallacy

- Very frequently, the reduction of number of defects is considered to be the ultimate goal of testing. However, it is not possible to assure that a software application, even if it has no defect, suits the needs of the public.

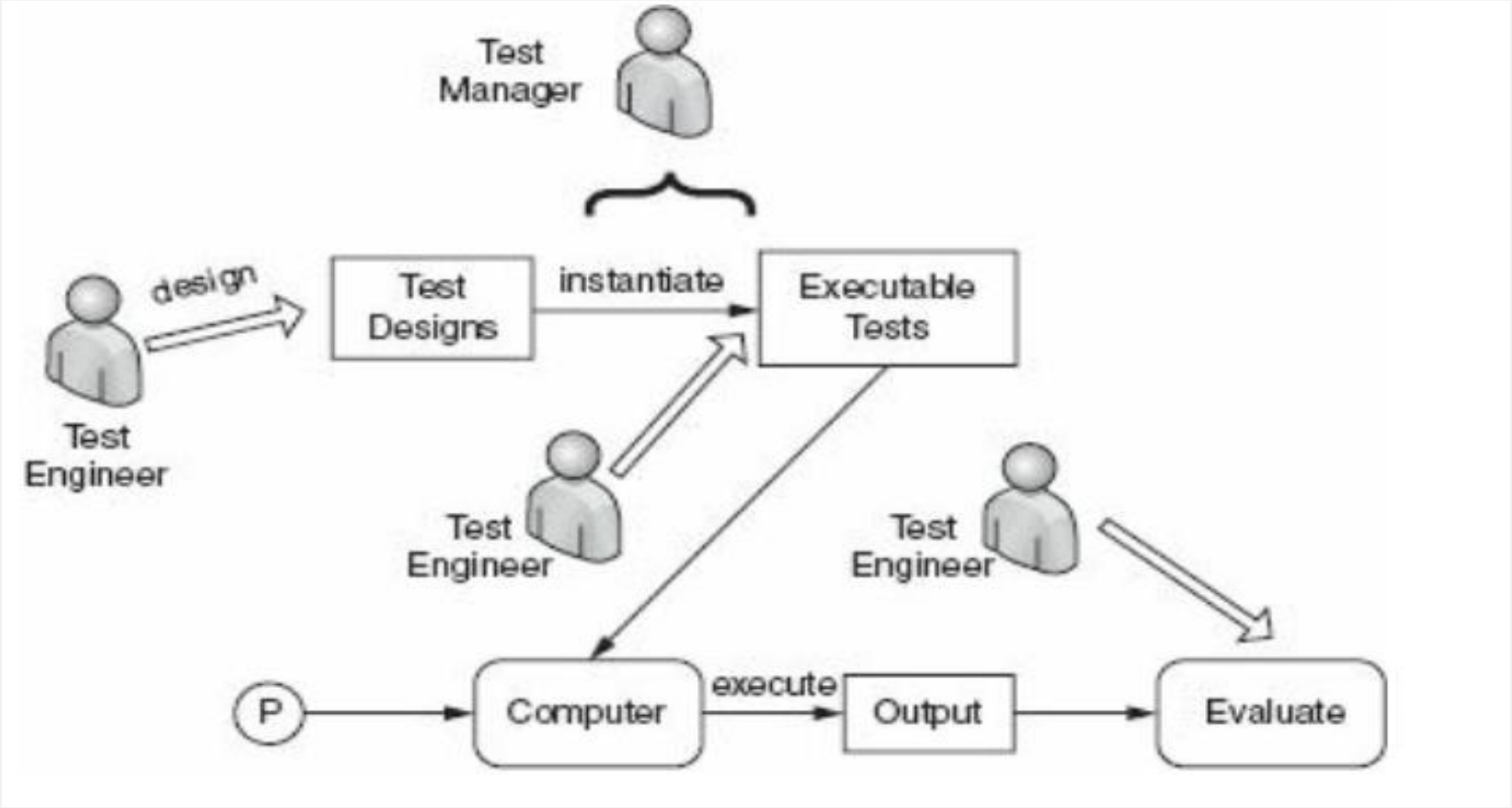
- If the software does not correspond to what the users need or to their expectations, identification and correction of defects will not help the architecture or the usability of the software, nor the financial success of the software.

Common terms in software testing

- Test Manager: Defines the scope of testing, manage resources required for testing, apply appropriate test measurements and metrics.

- Programmer: writes unit test code, designs system architecture and database.

- Test Engineer: prepares test plans, writes automated tests and performs functional testing.


Test
Manager
design
Test
Designs
instantiate
Executable
Tests
Test
Engineer
Test
Engineer
Test
Engineer
P
Computer
execute
Output
Evaluate

Exercises

1. Write a program that accepts the lengths of three sides of a triangle and determines the type of that specific triangle (isosceles, equilateral or scalene).

- İsosceles triangle: two sides equal
- Equilateral triangle: all sides equal
- Scalene triangle: none of the sides are equal

2. Identify the faults in the following program.

- Program to count the number of zeros in an array

```
Public static int counter (int[]givenArray)
{
int countZero;
for (int i=1:i<givenArray.length;i++)
If (givenArray(i)==0)
countZero=countZero+1;
}
return countZero;
```

Example, givenArray=[0,2,0,4,8,0]

https://www.cvedetails.com/top-50-products.php?year=2022