Software Testing

A study conducted by Microsoft and IBM showed that writing tests can add 15% – 35% to development time but reduce the number of bugs by 40% – 90%.

Software Testing

Testing verifies that the system meets the different requirements including, functional, performance, reliability, security, usability and so on.

This verification is done to ensure that we are building the system right.

In other words software testing is a **verification** and **validation** process.

What is Verification?

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase.
To make sure the product behaves the way we want it to.
It does not involve executing the code.
It is human based checking of documents and files.

What is Validation?

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase.
To make sure the product is built as per customer requirements.
It always involves executing the code.
It is computer based execution of program

SQA activities of Verification and validation ?
https://www.softwaretestinghelp.com/software-quality-assurance/

Testing Objectives

Testing is a process of executing a program with the intent of finding an error.

A good test case is one that has a high probability of finding an undiscovered error.

A successful test is one that uncovers an undiscovered error.

Characteristics of Testing

Testability: Software testability is simply how easily can be tested.

Operability: The better it works, the more efficiently it can be tested.

Observability: What you see is what you test.

Controllability. The better we can control the software, the more the testing can be automated and optimized.

Decomposability: By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting.

Simplicity: The less there is to test, the more quickly we can test it.

Stability: The fewer the changes, the fewer the disruptions to testing.

Understandability. The more information we have, the smarter we will test.

Testing Strategy

Any Testing strategy must incorporate:

Test planning
Test case design
Test execution
Resultant data collection and evaluation

Strategic Issues
Specify product requirements in a quantifiable manner long before testing commences.
State testing objectives explicitly.
Understand the users of the software and develop a profile for each user category.
Develop a testing plan that emphasizes “rapid cycle testing.”
Build “robust” software that is designed to test itself.

developer

Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

Must learn about the system,
but, will attempt to break it
and, is driven by quality

Types of Testing

There are many types of testing like:

- **Unit Testing**
- **Integration Testing**
- Functional Testing
- System Testing
- Stress Testing
- Performance Testing
- Usability Testing
- Acceptance Testing
- Regression Testing
- Beta Testing
- .....

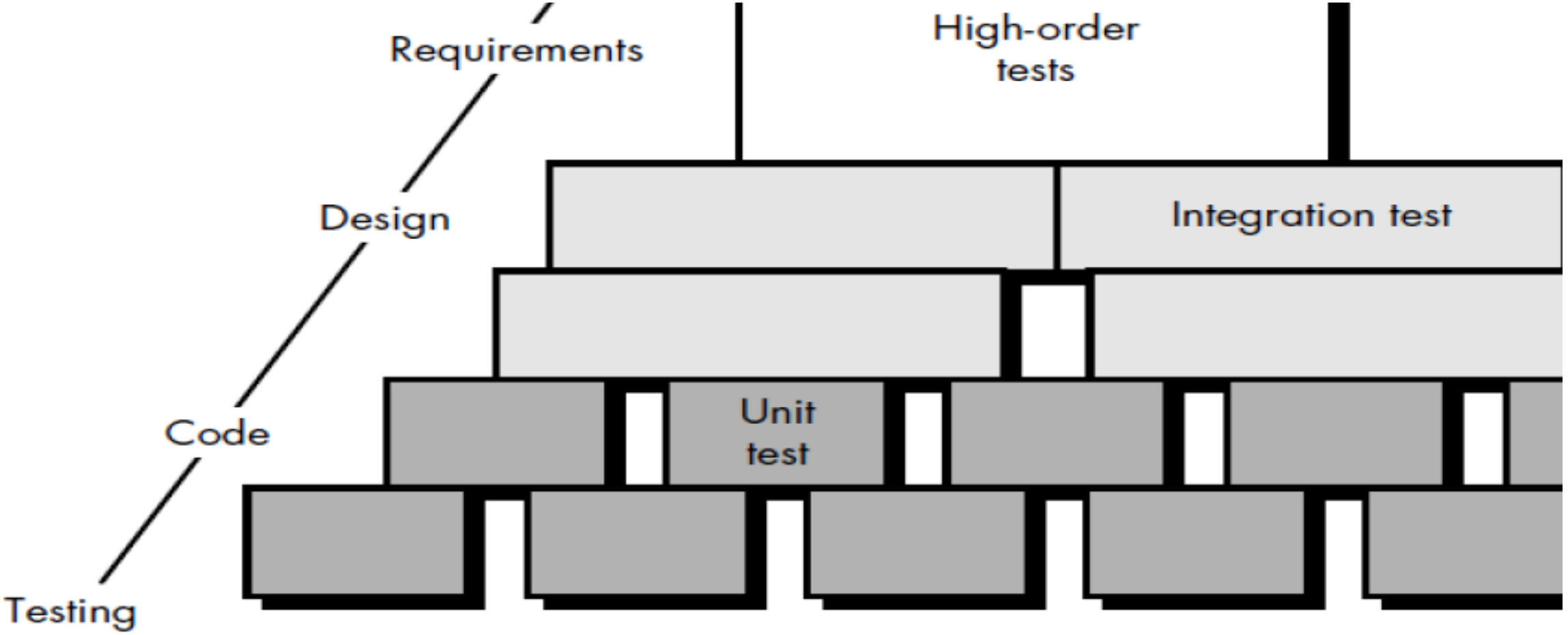
Unit Testing

Unit testing is that validate individual unit of source code working properly.
Unit is smallest testable part of an application.
Mostly done by developers of the modules.

Goal of Unit Testing

To isolate each part of program and show that individual parts are correct.

Procedural view of Testing Strategy



Advantage of Unit Testing.

Unit Testing reduces the level of bugs in production code.
Unit Testing inspires confidence.
Unit Testing makes it easier to change and refactor code.

Disadvantage of Unit Testing.

Unit-testing will not catch every error in the program.
By definition, it only tests the functionality of the Units.

Getting Started With Testing in Python

https://realpython.com/python-testing/

This is a great start for learning Testing in Python and should be read. This is also the source of the programs.

Programs.

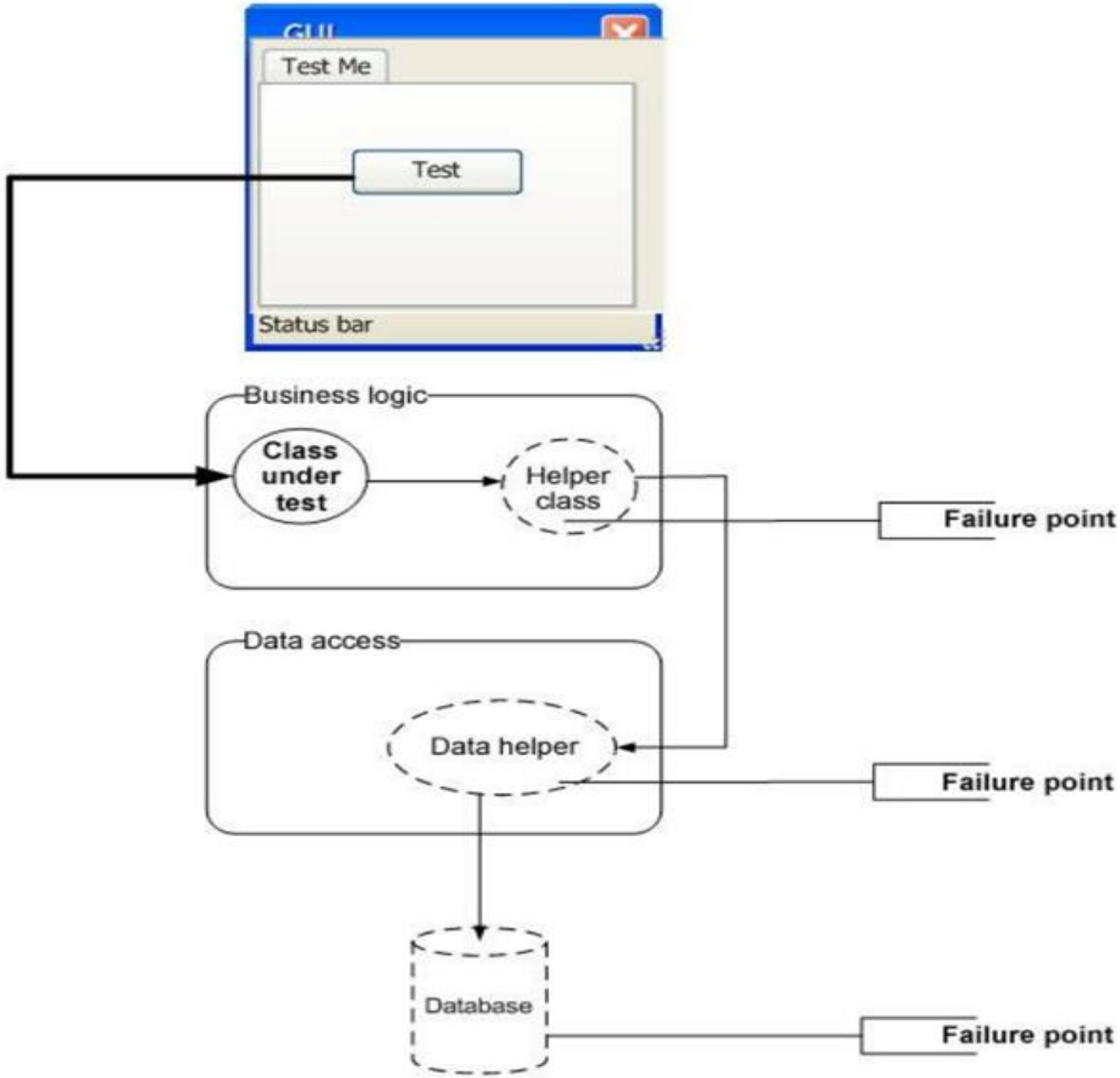
30 min to go through the programs.

INTEGRATION TESTING

Verifying software quality by testing two or more dependent software modules as a group.

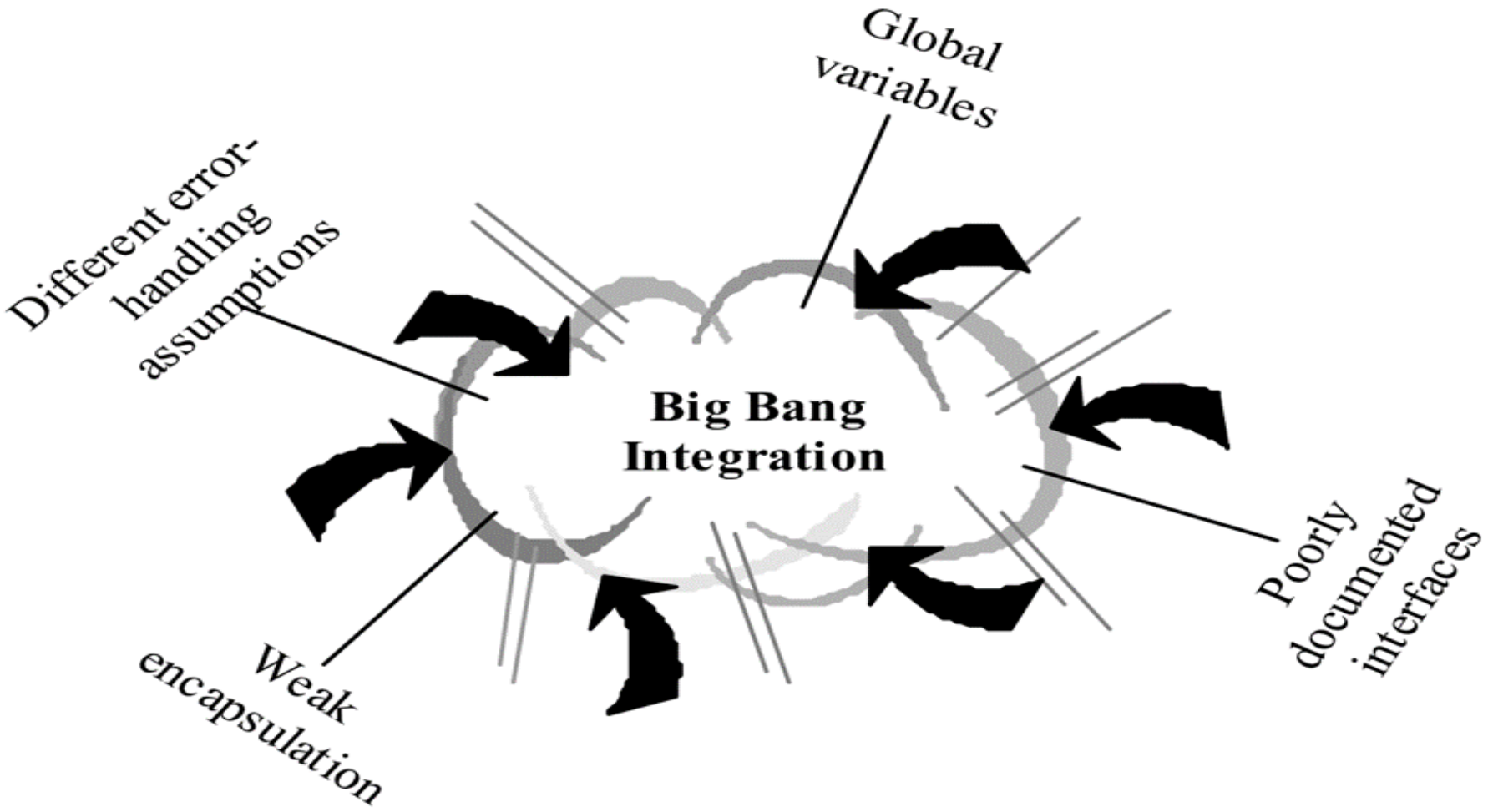
challenges:
Combined units can fail in more places and in more complicated ways.
How to test a partial system where not all parts exist ?
How to "rig" the behavior of modul A so as to produce a given behavior from modul B ?

New problems will inevitably surface:
Many systems now work together that have never been before.
If done poorly, all problems present themselves at once.
Cascade of interdependencies.


GUI
Test Me
Test
Status bar
Business logic
Class under test
Helper class
Failure point
Data access
Data helper
Failure point
Database
Failure point

Phased ("big-bang") integration:

Design, code, test, debug each subsystem separately and combine them all.
Pray.

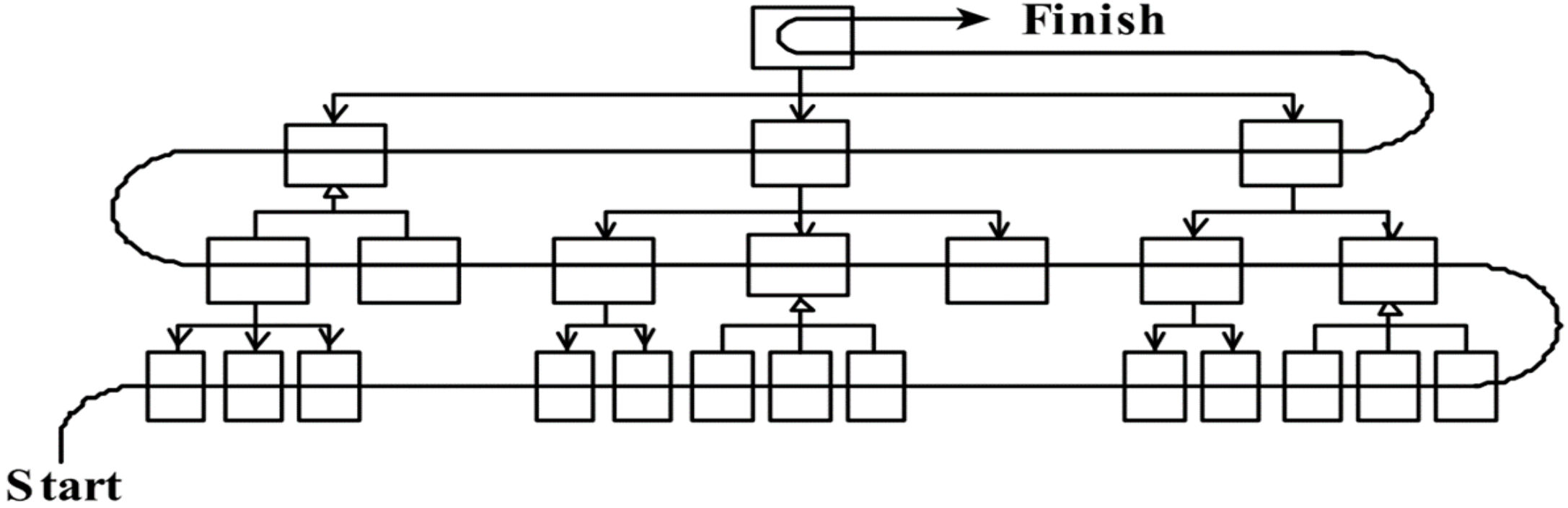


bottom-up integration:

Start with low-level data/logic layers and work outward.

Must write test drivers to run these layers.

Won't discover high-level / UI design flaws until late.



Exercise:

Your program must be tested in the form of Unit and Integration tests.
The following must be documented:

1. Test planning

2. Test case design.

3. Test execution.

4. Resultant data collection and evaluation.