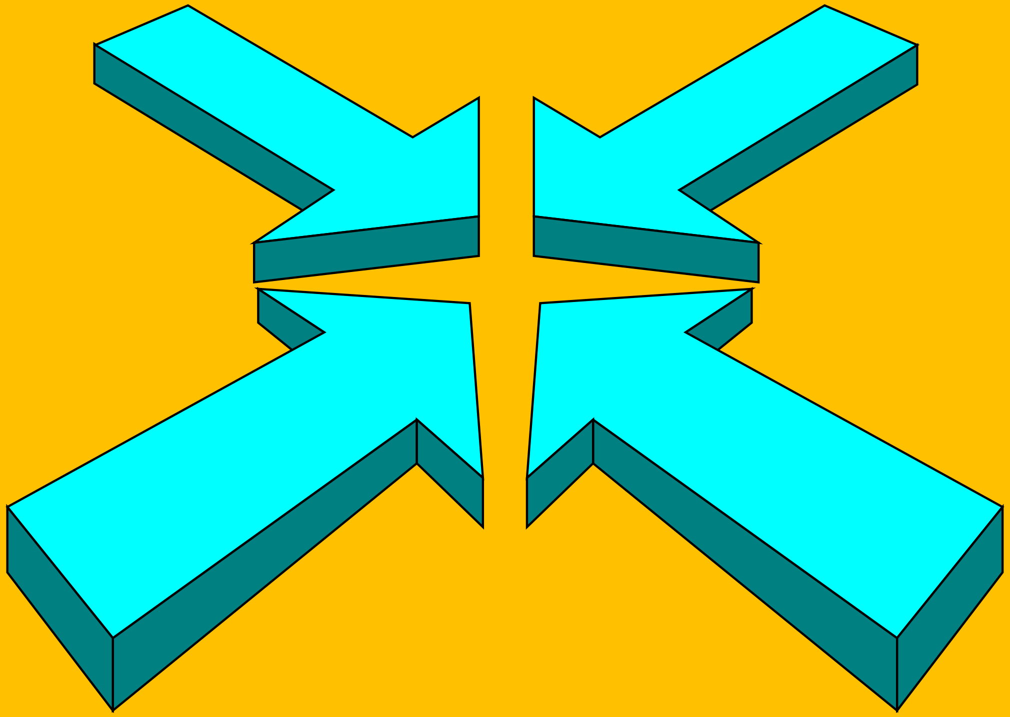
Software Testing Strategy

Testing Strategy



Unit test

Integration test

System test

Validation test

Testing Strategy

- Testing Strategy outlines in broad terms how to use testing to assess the extent to which the goal for a product has been met.
- Testing Strategy integrates software test case designing methods into a well planned series of steps that results in the successful development of a software.

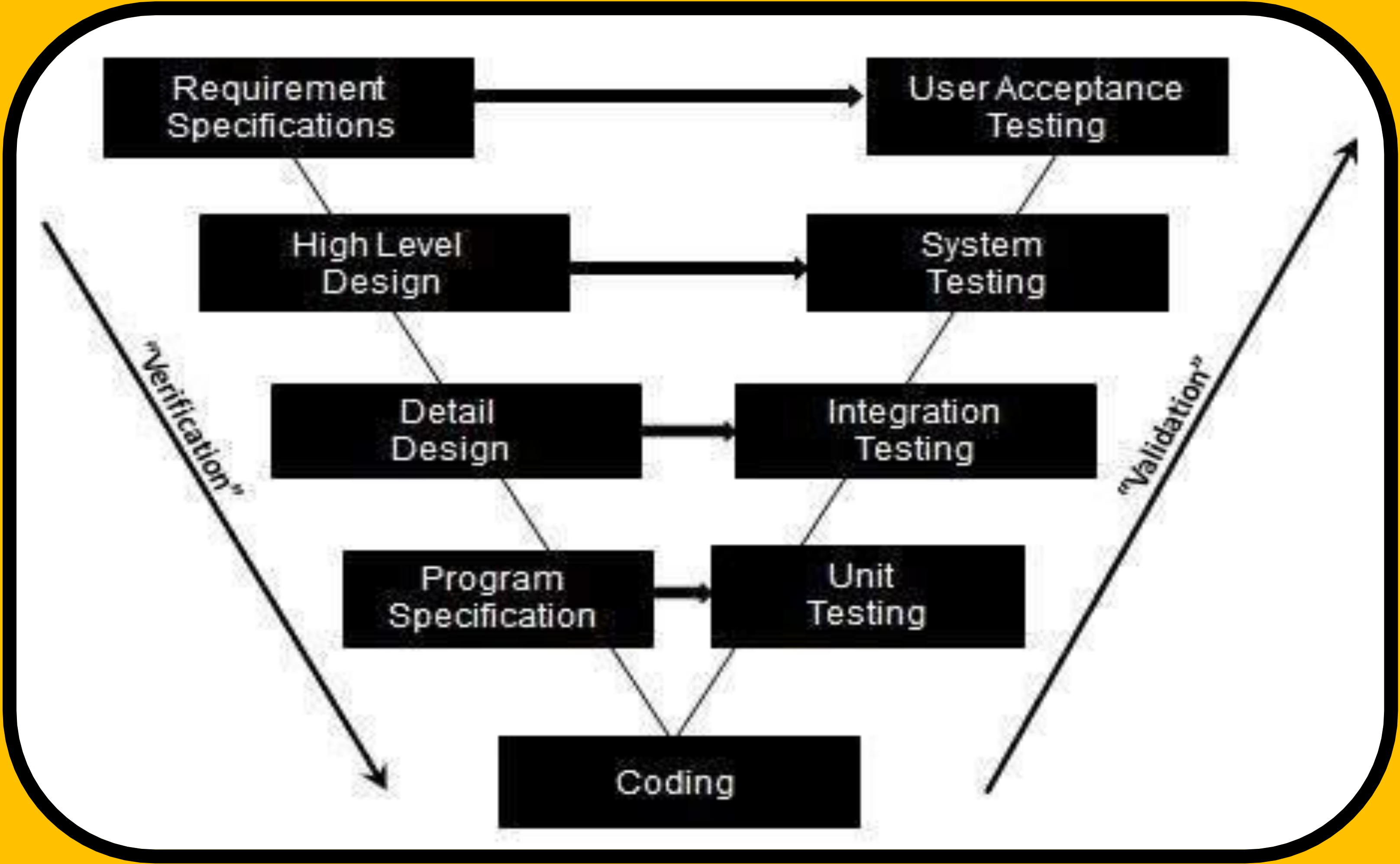
Testing Strategy Characteristics

- Testing begins at component level and works towards the integration of the entire system.
- Different testing techniques at different points of time.
- Testing is conducted by the developer and ITG.
- Testing and debugging are different, but debugging is accommodated in any testing strategy.

Why Testing Strategy?

- Testing often takes more project effort than any other software engineering activity.
- If it is conducted haphazardly, time is wasted, unnecessary effort is expended, and even worse, errors sneak through undetected.

The V-Model



Requirement Specifications
User Acceptance Testing
High Level Design
System Testing
Detail Design
Integration Testing
Program Specification
Unit Testing
Coding
"Verification"
"Validation"

The W-Model

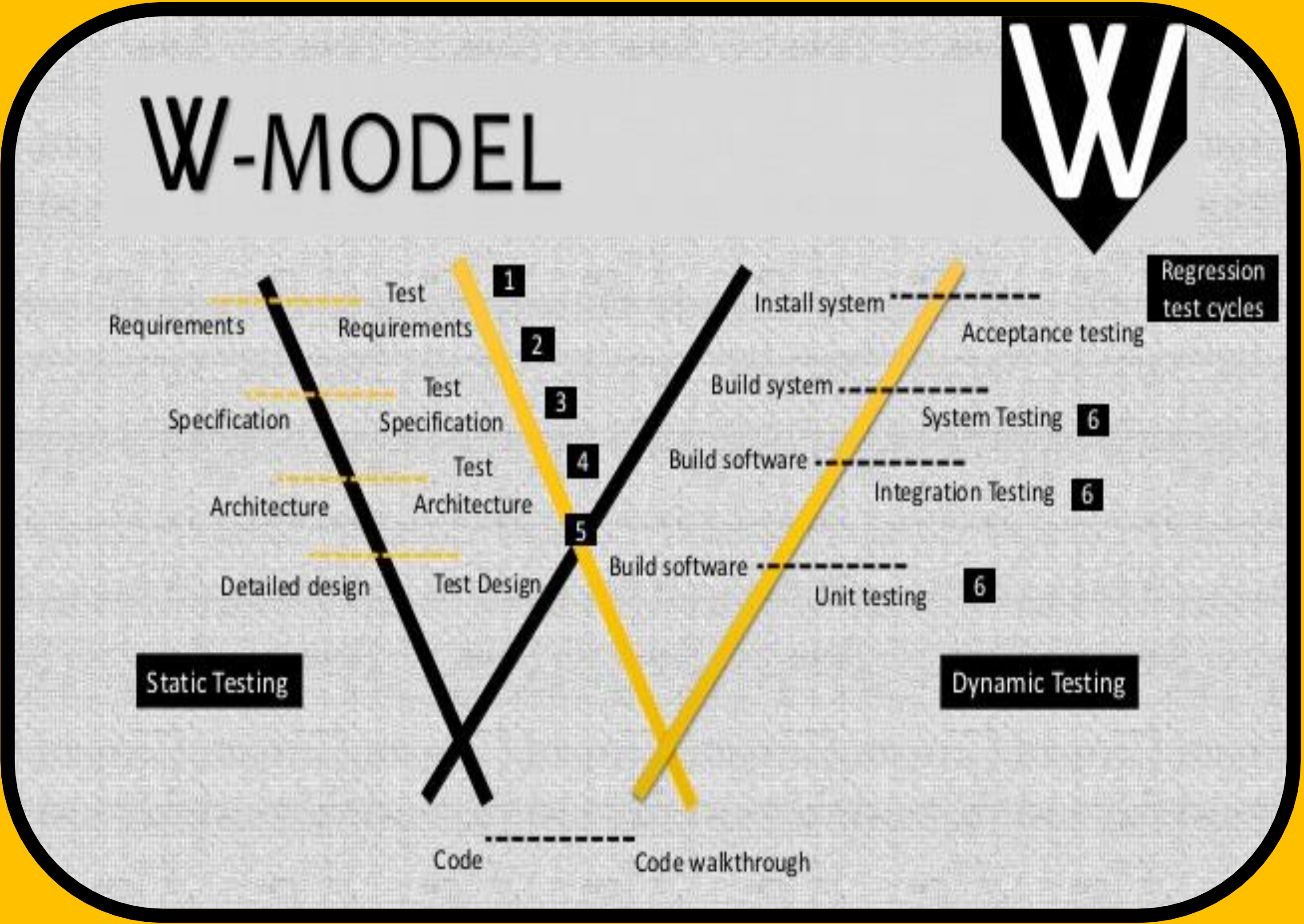
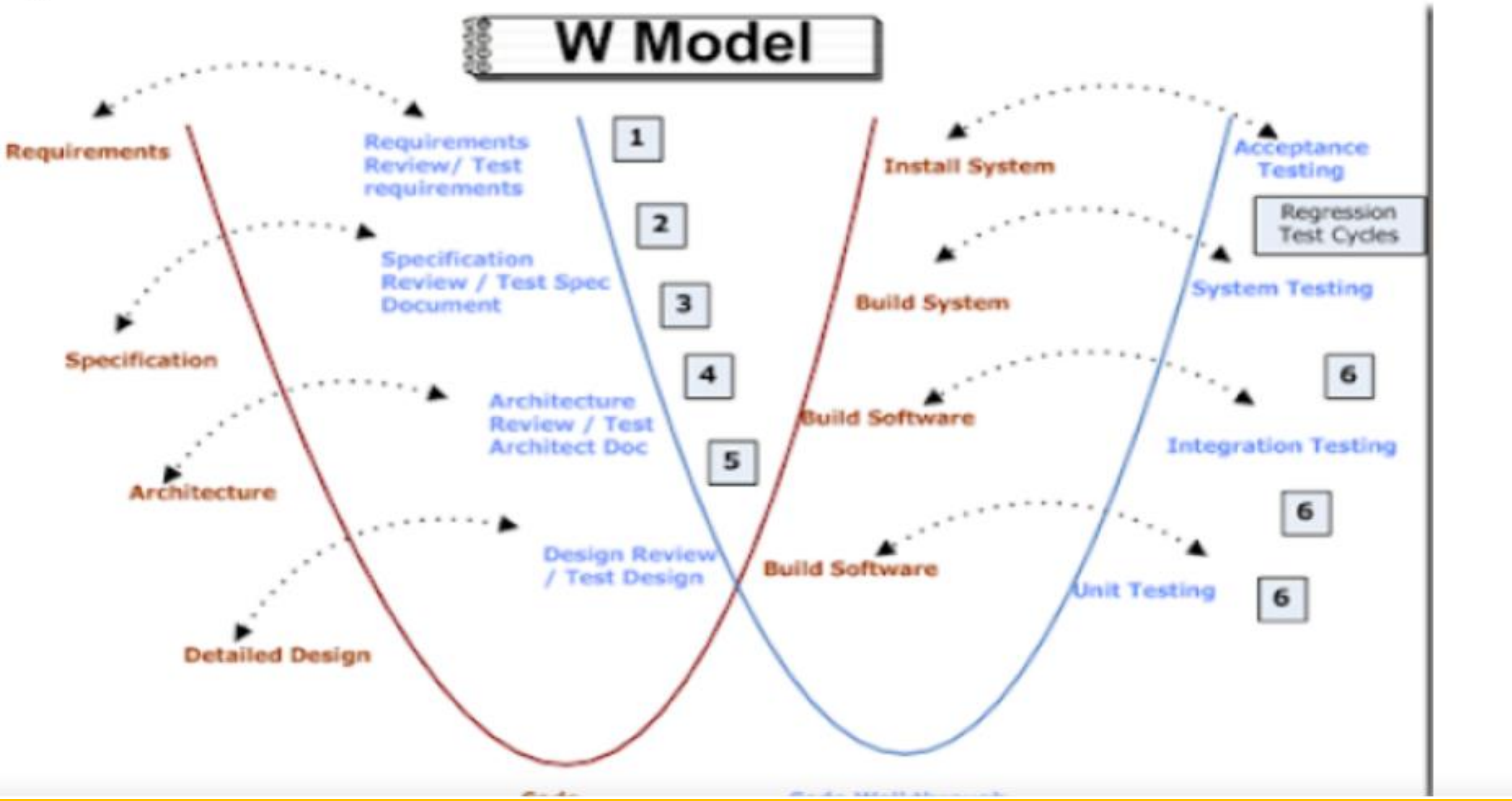
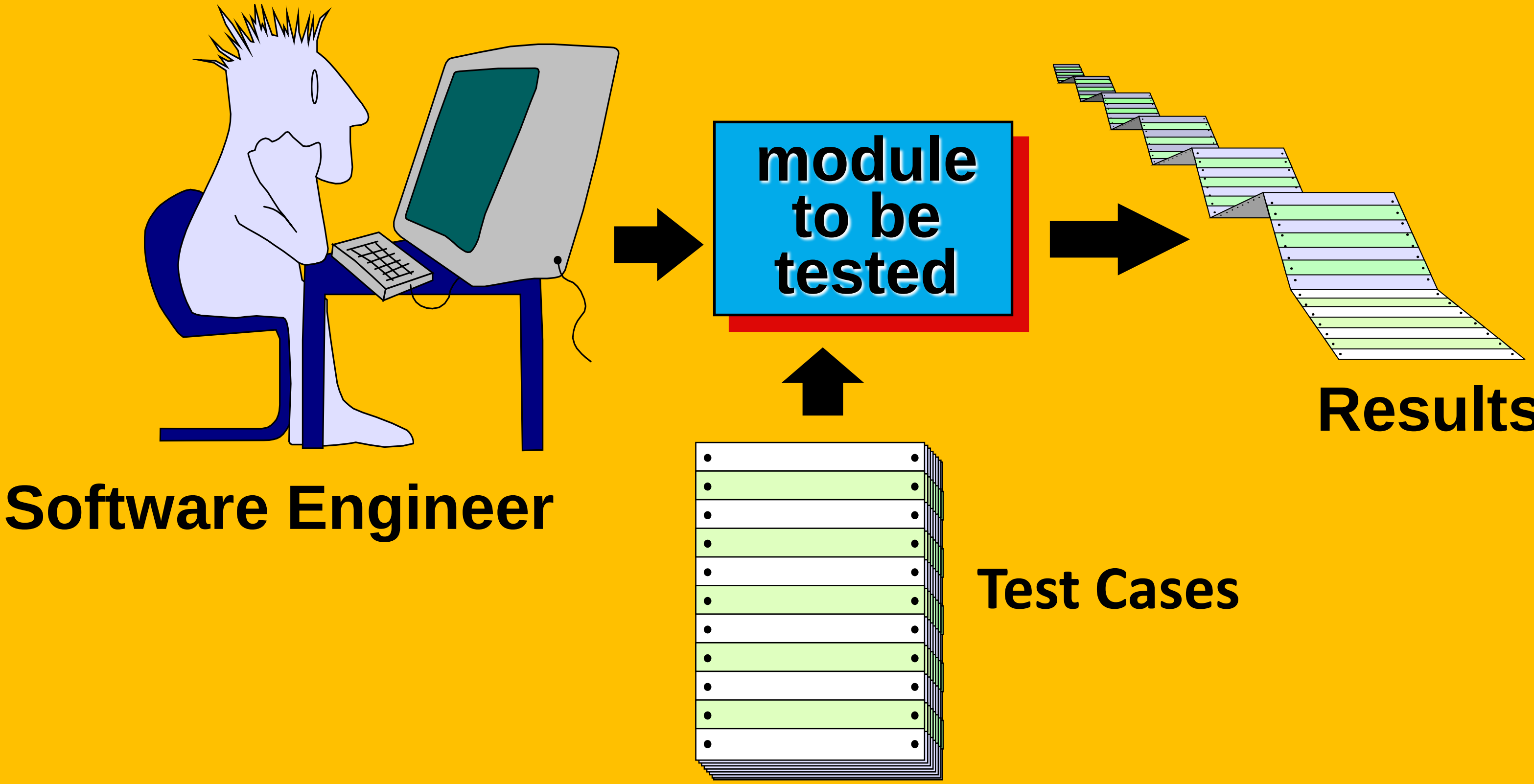

W-MODEL
Requirements
Test Requirements
1
2
Specification
Test Specification
3
Architecture
Test Architecture
4
5
Detailed design
Test Design
Code
Code walkthrough
Install system
Acceptance testing
Regression test cycles
Build system
System Testing
6
Build software
Integration Testing
6
Build software
Unit testing
6
Static Testing
Dynamic Testing

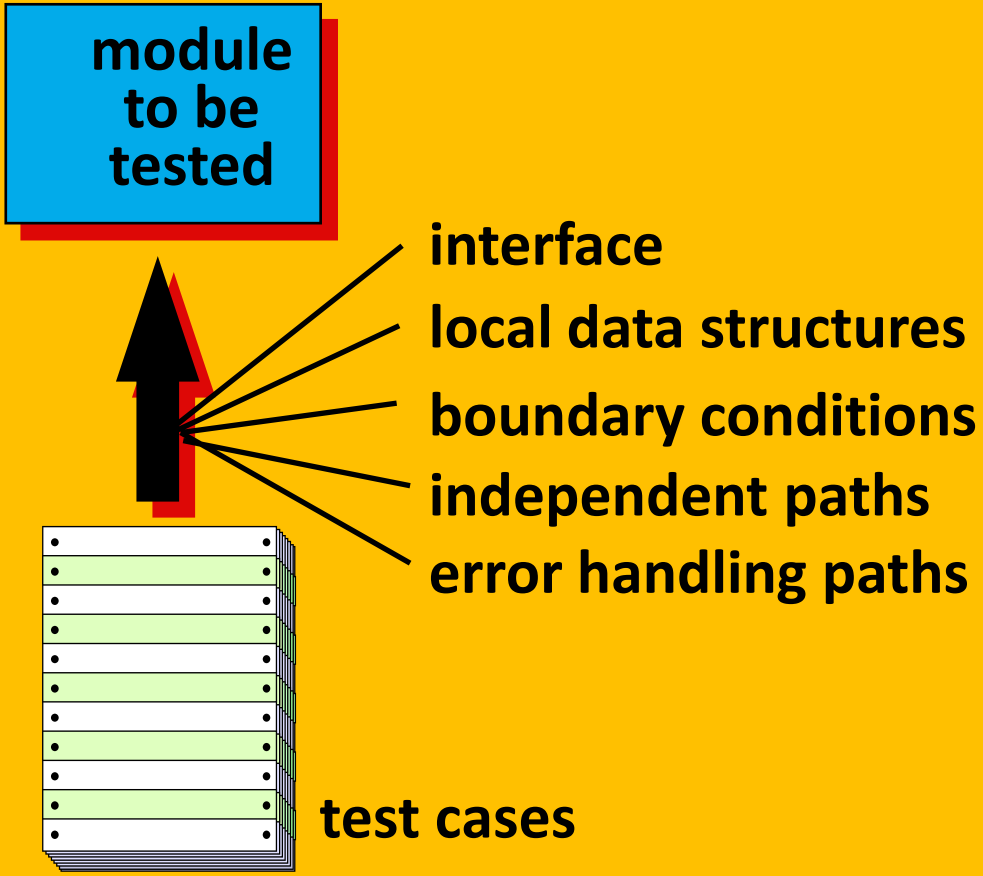
Fig 1: W Model


W Model
Requirements
Requirements Review/ Test requirements
1
Install System
Acceptance Testing
Regression Test Cycles
2
Specification Review / Test Spec Document
3
Build System
System Testing
Specification
4
Architecture Review / Test Architect Doc
Build Software
6
Integration Testing
5
Architecture
6
Design Review / Test Design
Build Software
Unit Testing
6
Detailed Design

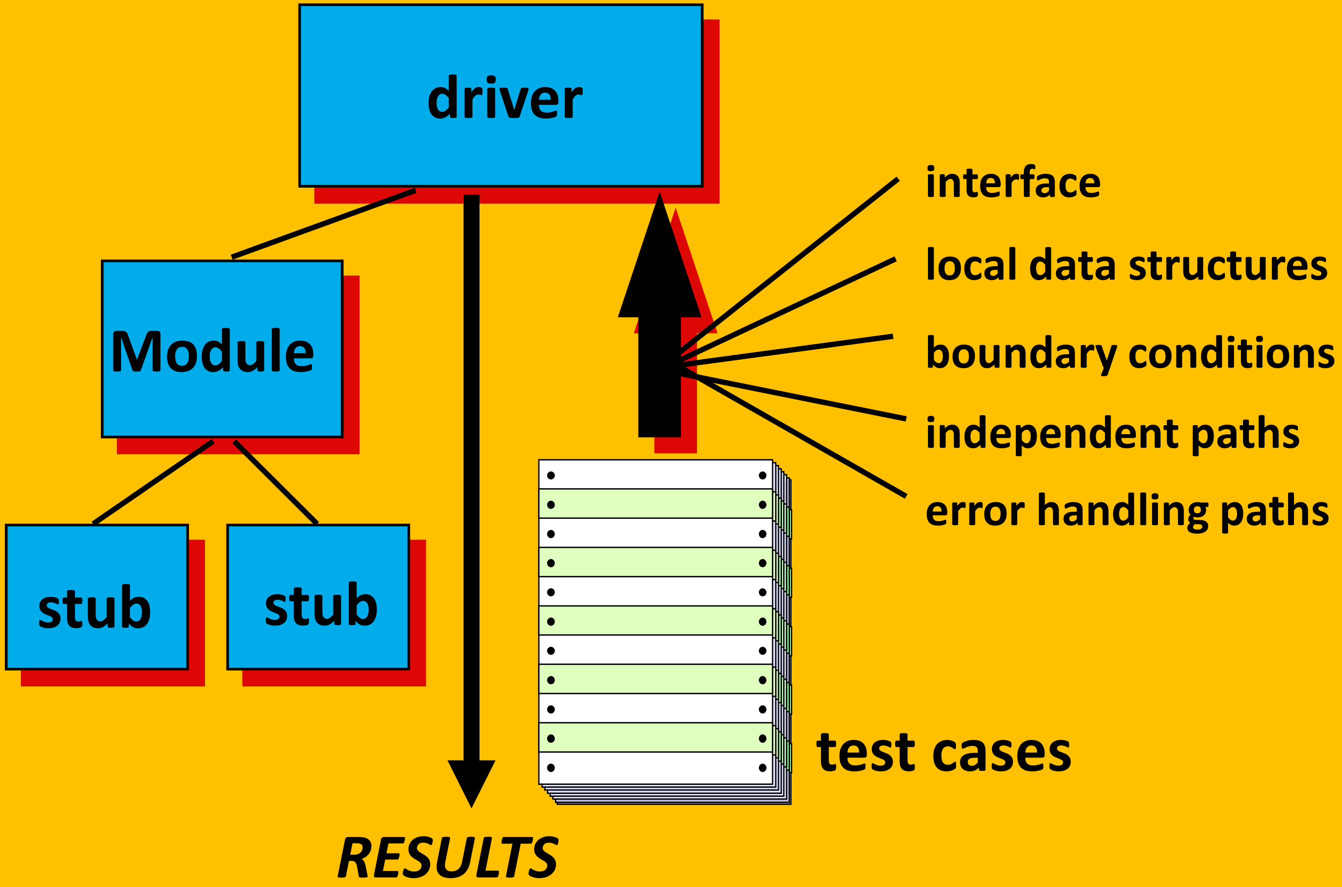
Unit Testing



Unit Testing



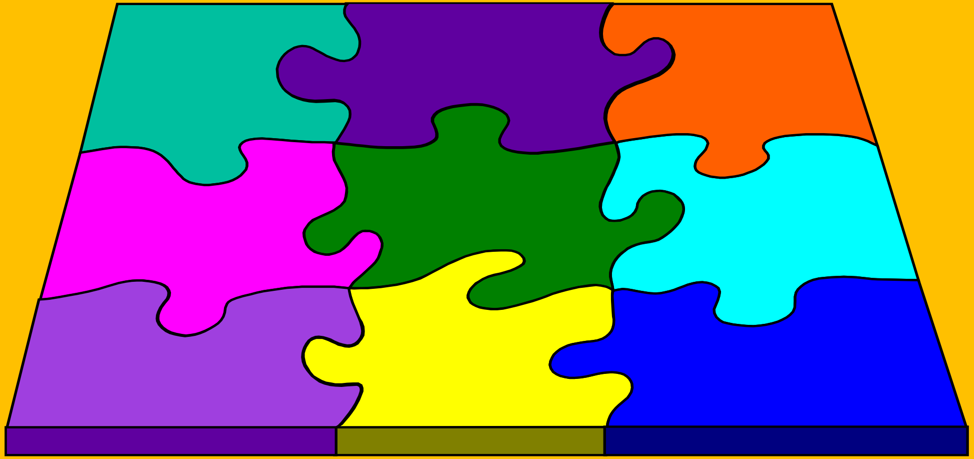
Unit Test Environment



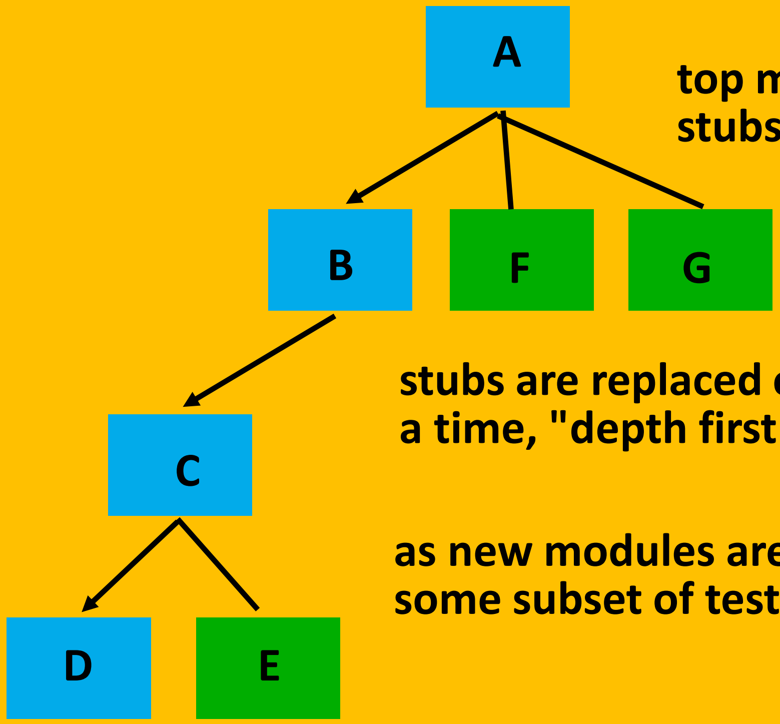
Integration Testing Strategies

Options:

- **The “big bang” approach**
- **An incremental construction strategy**



Top Down Integration



top module is tested with stubs

stubs are replaced one at a time, "depth first"

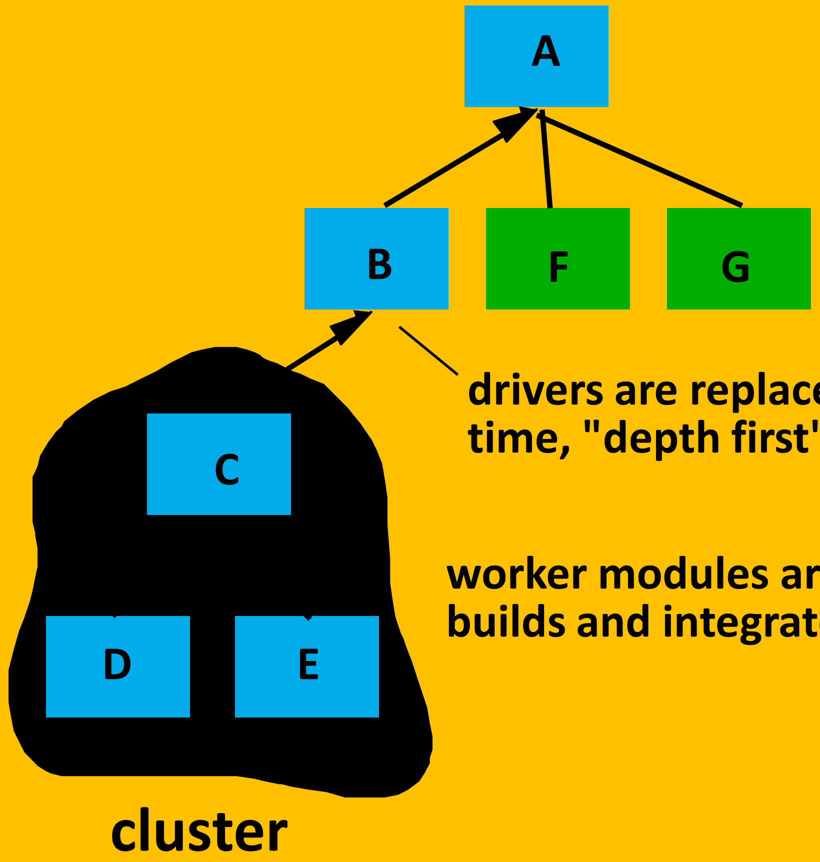
as new modules are integrated, some subset of tests is re-run

Steps for integration process

1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module.
2. Depending on the integration approach selected subordinate stubs are replaced one at a time with actual components.
3. Tests are conducted as each component is integrated.
4. On completion of each set of tests, another stub is replaced with the real component.
5. Regression testing may be conducted to ensure that new errors have not been introduced.

The process continues from step 2 until the entire program structure is built.

Bottom-Up Integration



drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

Steps for Bottom Up Integration

1. Low level components are combined into clusters that perform a specific software sub function.
2. A driver is written to coordinate test case input and output
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structures

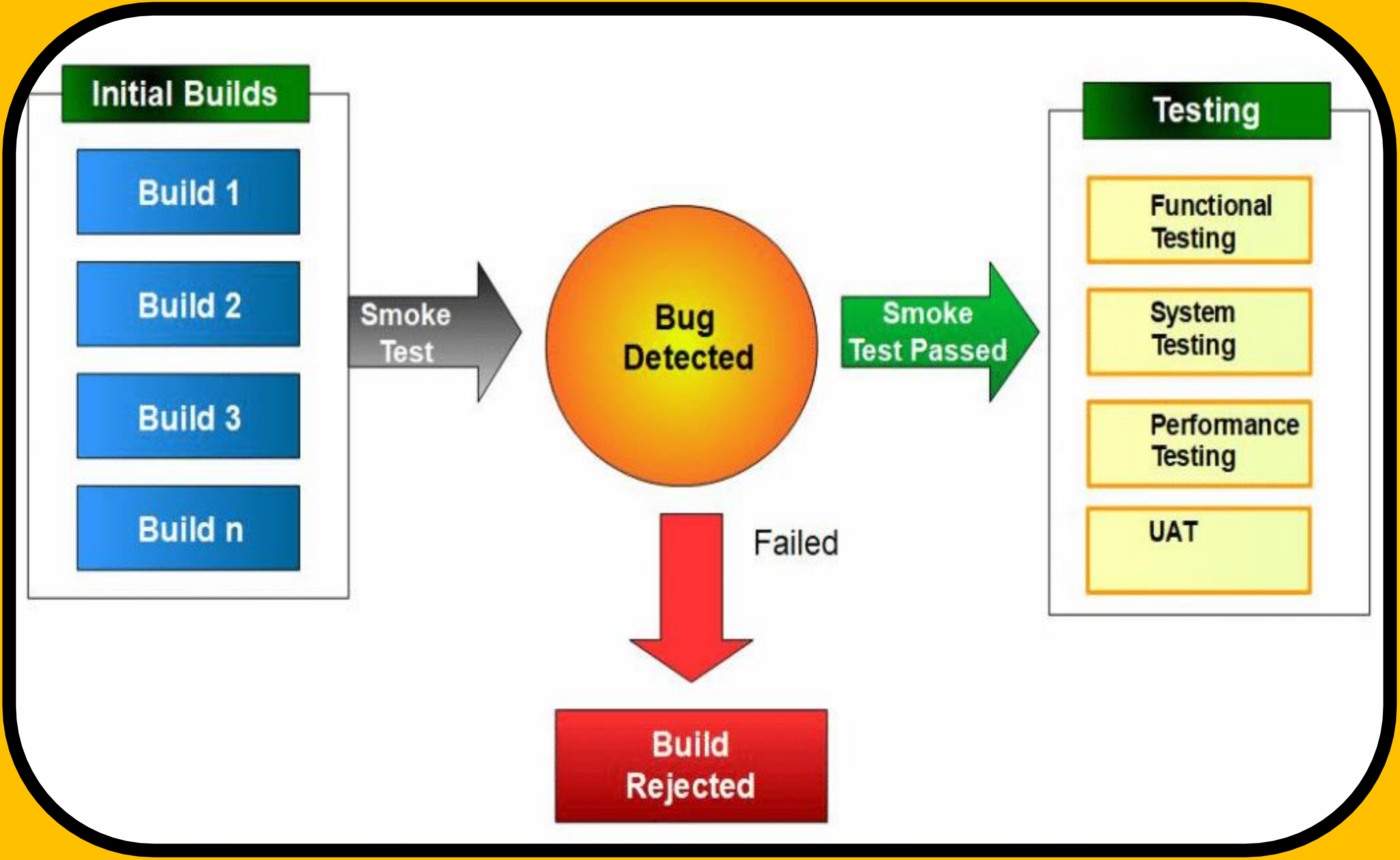
Smoke Testing

Used for "Shrink Wrapped" software and time – critical project

Activities Included in Smoke testing:

1. Software Components are integrated into build
2. A series of tests is designed to expose errors that will keep the build from properly performing functions
3. The build is integrated with other builds and the entire product is smoke tested daily. The integration approach may be top down or bottom up.

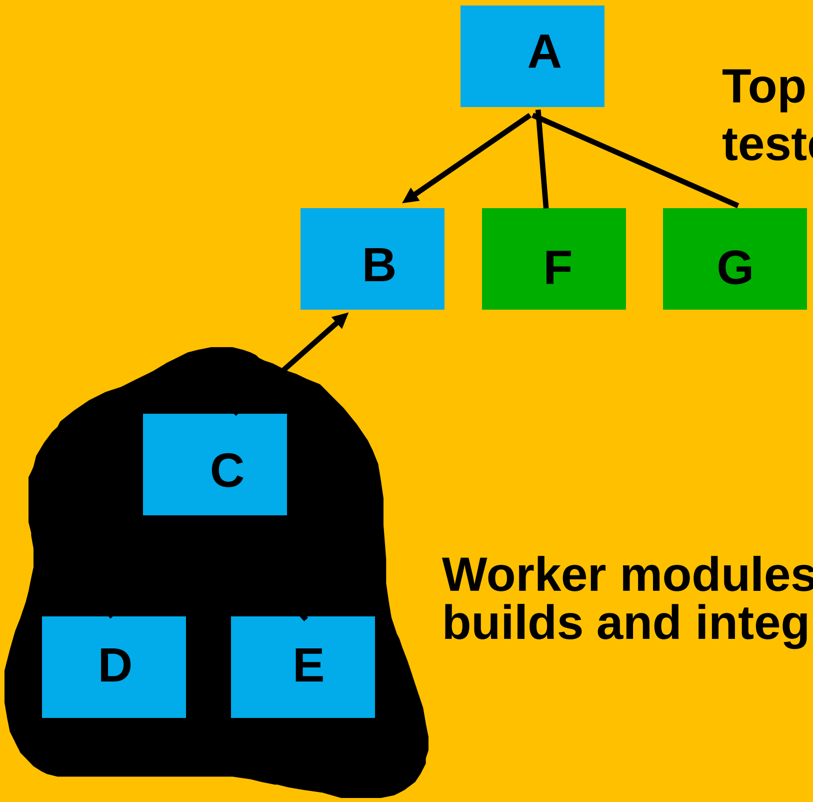
Smoke Testing



Comments on Integration Testing

1. Stub problem
2. Sandwich testing
3. Critical Module

Sandwich Testing



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

Validation Testing

Validation:

Validation succeeds when software functions in a manner that can be reasonably expected by the customer.

Validation Criteria:

- Test plan which conforms the user requirements

Consequences are tackled accordingly.

“Are we building the product right”--------(Verification)
“Are we building the right product”----------(Validation)

Alpha and Beta Testing

- **Alpha Testing:**

  AT is conducted at the developer's sites by the customer in a controlled environment.

- **Beta Testing:**

  The beta test is conducted at one or more sites by the end-user of the software

System Testing

- Testing of the software when it interacts with hardware, people, and information.
- These tests are out of the scope of the software

Types of system Testing:

1. Recovery Testing
2. Security Testing
3. Stress Testing
4. Performance Testing

Recovery Testing

- RT is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.

Security Testing

- ST attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.

Stress Testing

- Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.

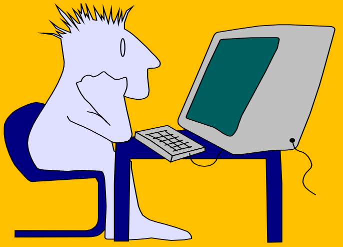
Performance Testing

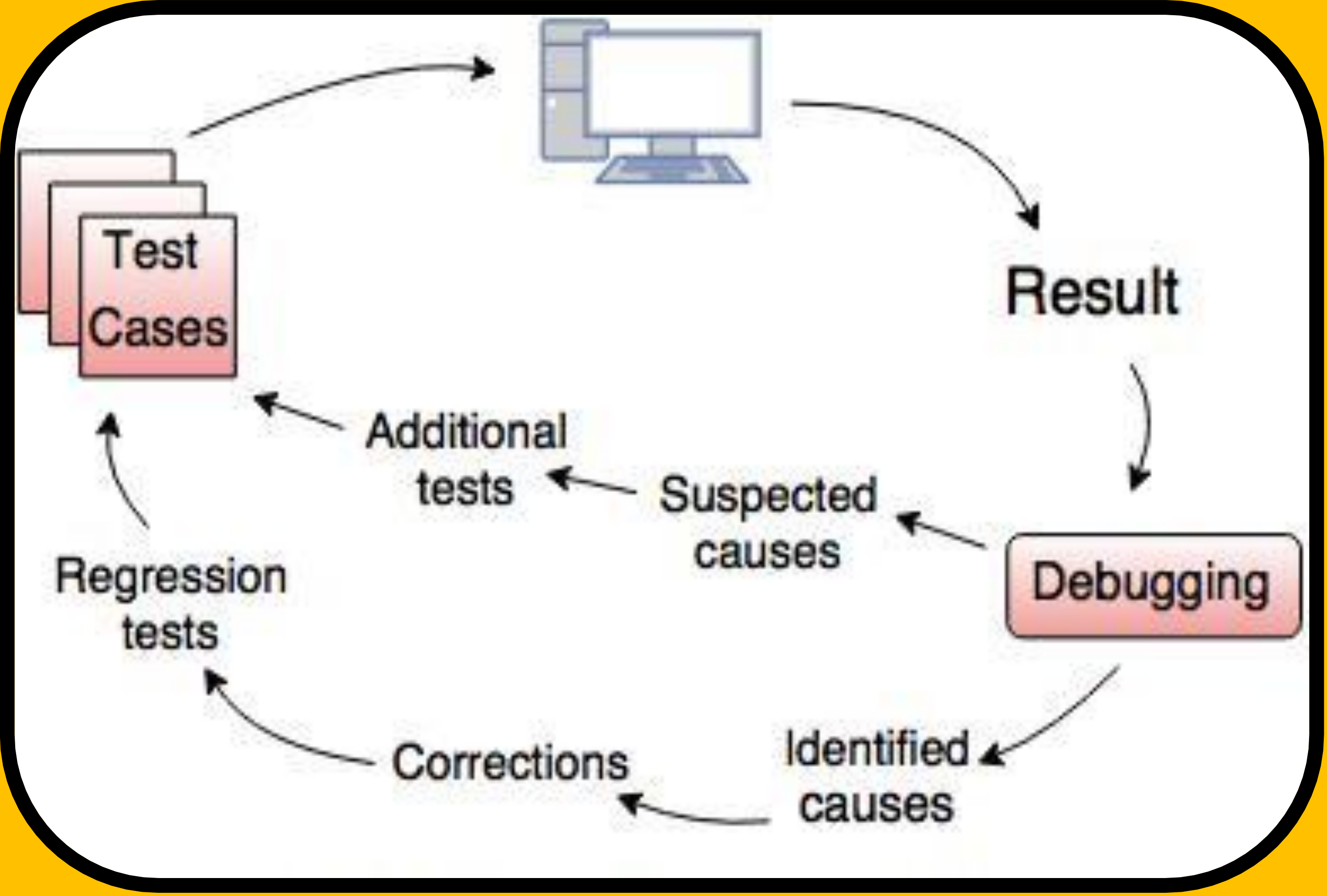
- PT is designed to test the run-time performance of software within the context of an integrated system.

Debugging: A Diagnostic Process

Debugging occurs as a consequence of successful testing. That is, when a test case uncovers an error, debugging is the process that results in the removal of the error.

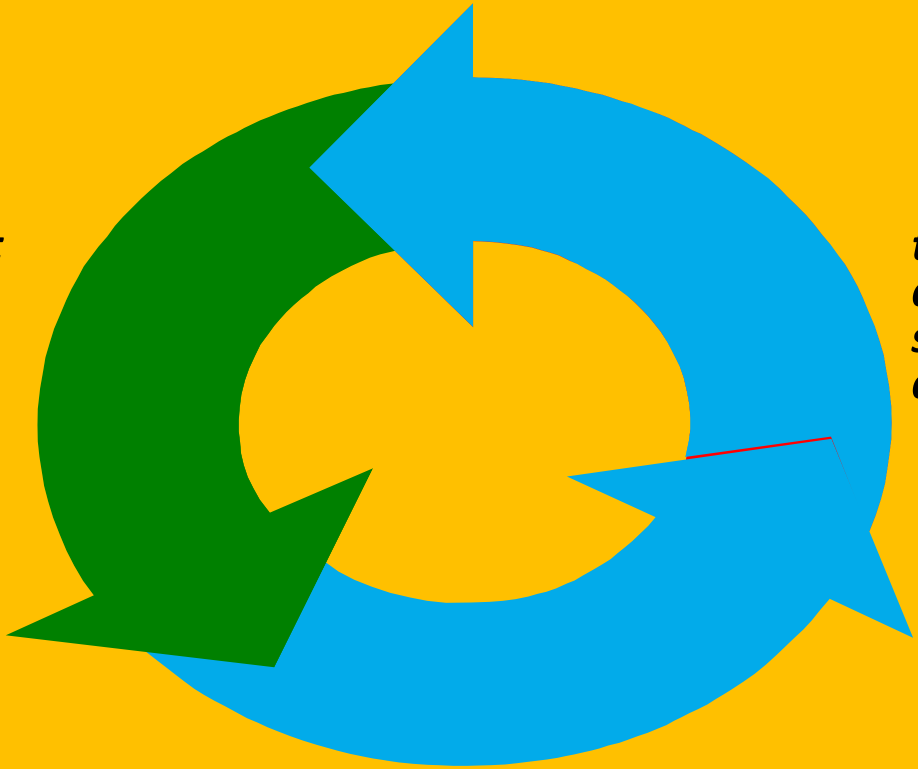
Debugging Process:





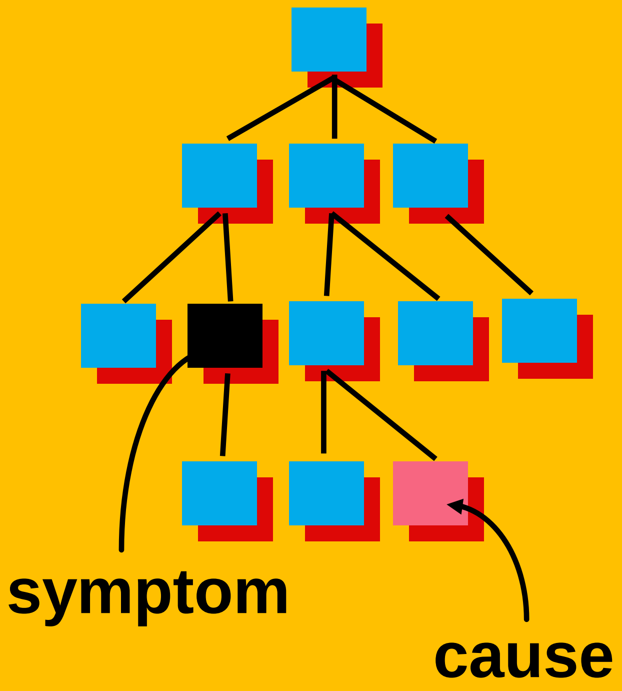
Debugging Effort

time required to correct the error and conduct regression tests



time required to diagnose the symptom and determine the cause

Symptoms & Causes



- symptom and cause may be geographically separated
- symptom may disappear when another problem is fixed
- cause may be due to a combination of non-errors
- cause may be due to a system or compiler error
- cause may be due to assumptions that everyone believes

Debugging Techniques

- brute force / testing
- backtracking
- cause elimination

Debugging: Final Thoughts

1. Don't run off half-cooked, think about the symptom you're seeing.

2. Use tools (e.g., dynamic debugger) to gain more insight.

3. If at an impasse, get help from someone else.

4. Be absolutely sure to conduct regression tests when you do "fix" the bug.

Thank You!