Grand Valley State University
www.gvsu.edu

Software Engineering Verification (and Validation)

Erik Fredericks // frederer@gvsu.edu

Adapted from materials provided by Byron DeVries, Jagadeesh Nandigam

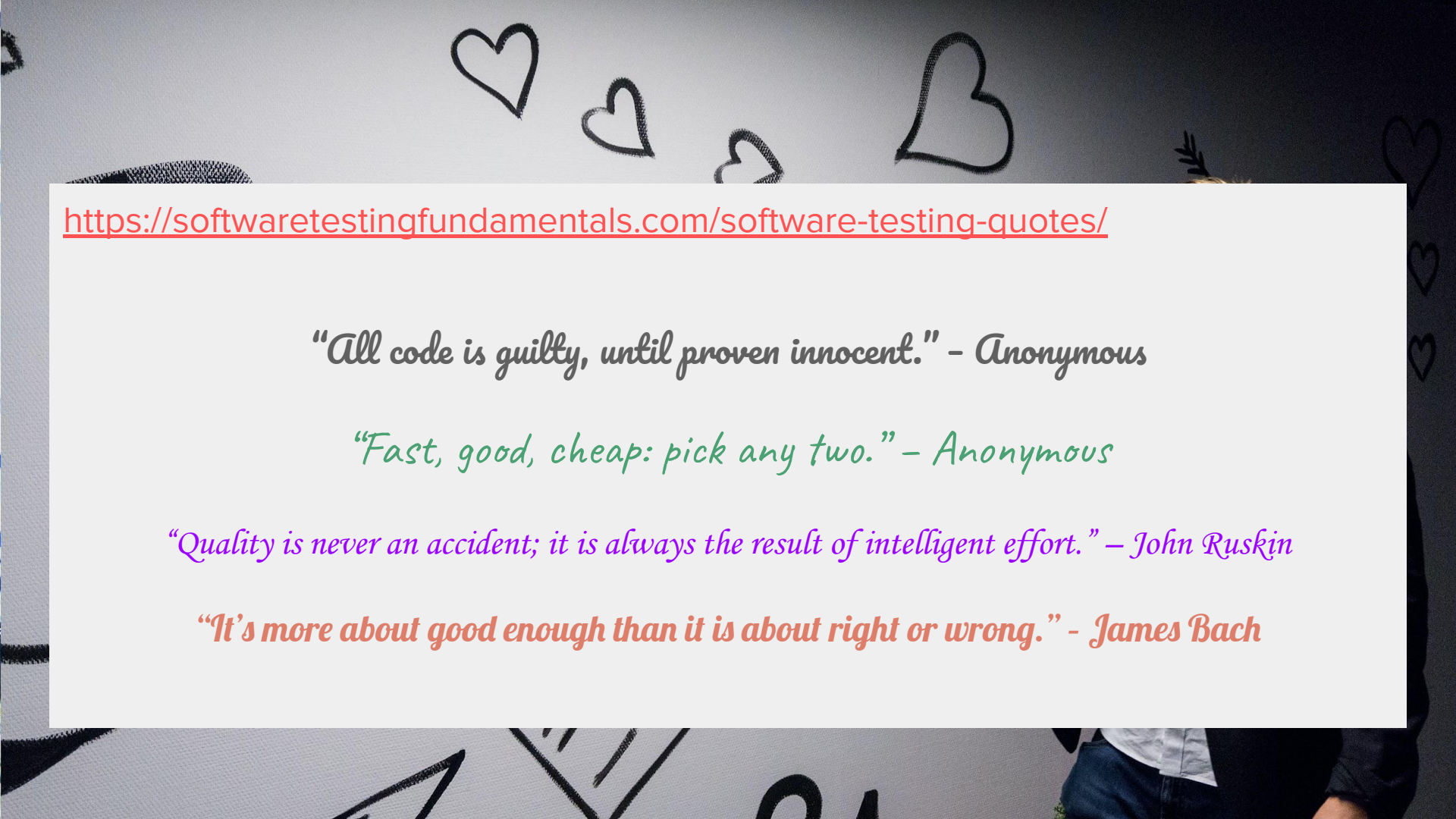
Verification vs Validation

Types of Verification

- Dynamic Verification
- Static Verification

Software Testing

- Definition
- Psychology and Economics
- Testing Processes
- Testing Levels



https://softwaretestingfundamentals.com/software-testing-quotes/

"All code is guilty, until proven innocent." – Anonymous

"Fast, good, cheap: pick any two." – Anonymous

"Quality is never an accident; it is always the result of intelligent effort." – John Ruskin

"It's more about good enough than it is about right or wrong." - James Bach

Verification vs. validation

Verification: A set of activities that ensure that the software conforms to its specifications:

“Are we building the thing right?”

Validation: A set of activities that ensure that the software built meets the customer’s needs and expectations.

“Are we building the right thing?”

Verification and Validation (V&V) should establish confidence that the software is “fit for purpose”

➜ This does not mean the software is completely free of defects.

➜ It does mean the software must be good enough for its intended use, where that use determines the degree of confidence that is needed.

Types of verification

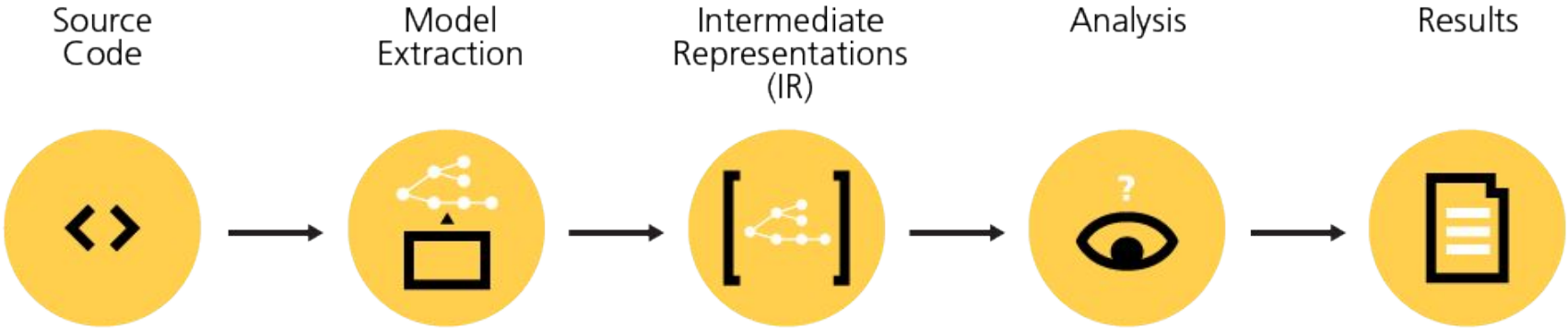
Dynamic Verification (e.g., software testing):

- Software implementation is executed with test data.
- Applied to a prototype or an executable version
  - Actual vs expected outputs are compared.

Static Verification:

- Analyze and check system representations such as requirements, specifications, design, and source code.
- Can be applied at all stages of the software process
- Walkthrough, inspections, desk checking, correctness proofs, and automated static analyzers.

(next slide ➜ static analysis)

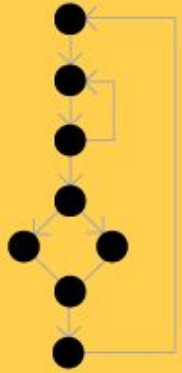


Names Databases/Symbol Table

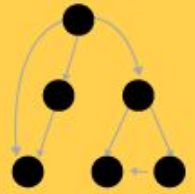
Name	Kind	Location
copy_item	function	item. c:25
item_cache	variable	itemc:10
color	parameter	pallette.c:23
header.h	file	chapes.c

Abstract Syntax Tree (AST)

Control Flow graph (CFG)



Call Graph



Verification vs Validation

Can **verification** activities be used for **validation**?

Software testing

Testing is the process of executing a program with the intent of finding errors*.

What is a *successful* test?

Commonly, we *think* of a successful test as a *passing* test. However…

A *successful* test one that establishes the presence of *one or more* errors in the software.

Glenford Myers, *The Art of Software Testing*

Software testing

Testing can only demonstrate the **presence of errors** in a program, not their absence.

- Simply because a test suite (i.e., a collection of tests) for a program doesn't detect any errors (i.e., failing tests), doesn't mean the program is error free.
- It could easily be that the test suite never exercised, or tested, the program in such a way to indicate that there is an error.
- Realistically, large programs are never* bug free.

** or very rarely*

Psychology of testing

Proper (and correct) definition of testing depends upon the psychological attitude of the person doing the testing.

Destructive nature of testing:

- Testing is intended to cause a program to behave in a manner it was not intended or thought of by its designer or programmer.
- Psychologically, programmers do not want to destroy their creations. They tend to select tests that will not demonstrate the presence of errors.
- A programmer should avoid attempting to test his or her own program.

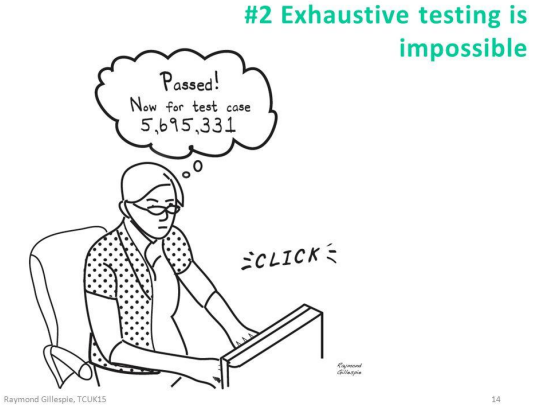
How does this compare to Test-Driven Development?

Economics of testing

Forms of exhaustive testing:

- Exhaustive input (Black-Box) testing
- Exhaustive path (White-Box) testing

Is exhaustive testing feasible?



No, exhaustive testing is impractical and often impossible. Therefore, systematic testing techniques are needed.

Exhaustive Testing - Example

How long would it take (approximately) to test exhaustively the following program?

```
int sum(int a, int b) {return a + b;}
```

1. $2^{32} \times 2^{32} = 2^{64} =\sim 10^{19}$ tests
2. Assume 1 test per nanosecond (10^9 tests/second)
3. We get 10^{10} seconds…
4. About 600 years!

Objective of testing

In most cases, testing is conducted in an ad hoc fashion which occurs late in a software project, when time is short, budgets are nearly exhausted, managers are nervous, and customers are anxious.

Testing strategies and techniques enable us to improve the thoroughness with which software tests are conducted and increase the probability that testing will achieve its primary objective:

“to find the largest number of errors in the least amount of time with a minimum allocation of resources”

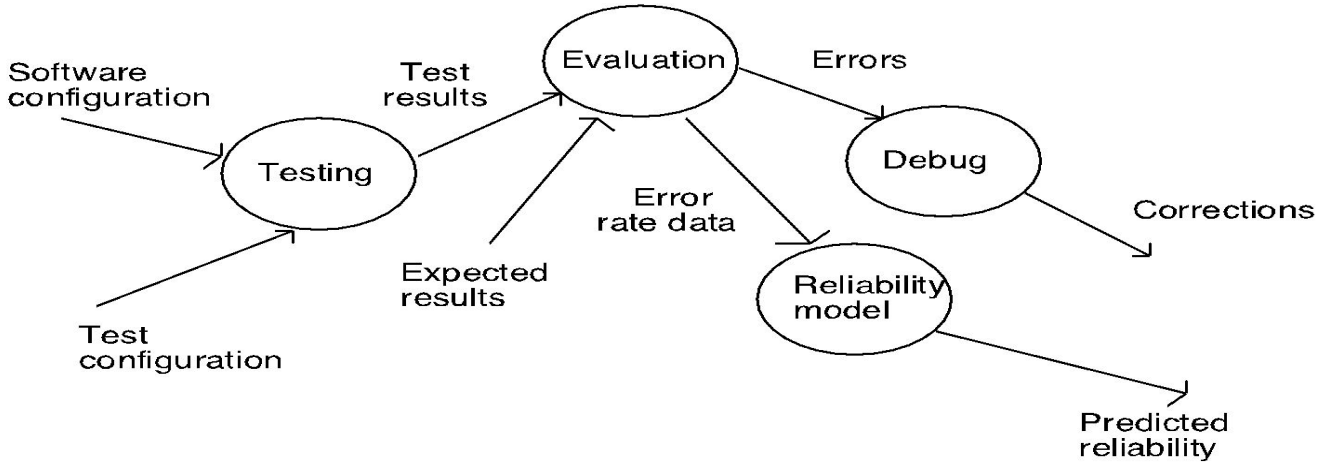
Software testing process

Software Configuration:

- Requirements specifications, design specifications, and source code

Test Configuration:

- Test plans, procedures, and tools
- Test cases (test data and expected results)



Testing guidelines

Testing Guidelines / Principles:*

1. A necessary part of a test case is a definition of the expected output or result.
2. A programmer should avoid attempting to test his or her own program.
3. A programming organization should not test its own programs.
4. Thoroughly inspect the results of each tests.
5. Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected.

**As defined by Glenford J. Myers in The Art of Software Testing*

Testing guidelines cont'd.

Testing Guidelines / Principles:*

6. Examining a program to see if it does not do what it is supposed to do is only half the battle; the other half is seeing whether the program does what it is not supposed to do.

7. Avoid throwaway test cases unless the program is truly a throwaway program.

8. Do not plan a testing effort under the tacit assumption that no errors will be found.

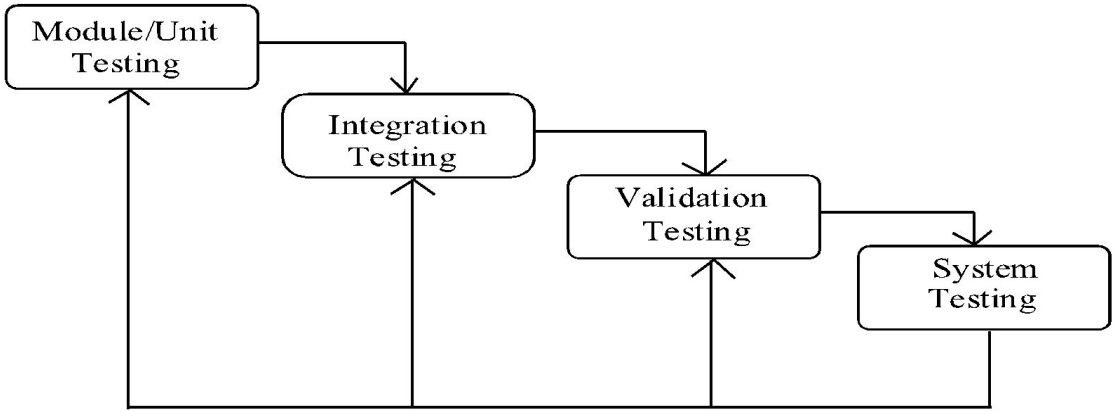
9. The probability of the existence of more errors in a section of a program is proportional to the number of errors already in that section.

10. Testing is an extremely creative and intellectually challenging task.

Testing levels

Testing Strategy: Incremental view of testing:

Begin with testing of individual program units, moving to tests designed to facilitate the integration of the units, and culminating with tests that exercise the constructed system.



Unit testing

Unit/module/component testing:

- Focuses verification effort on the smallest unit of software design.

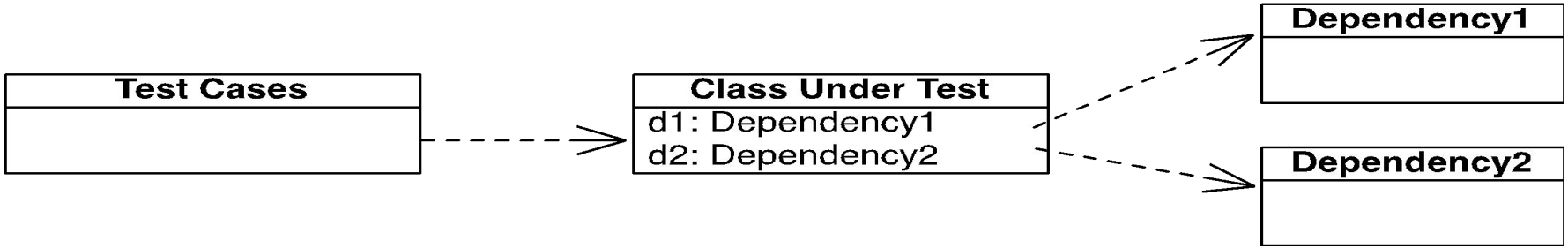
Unit testing is normally considered as **secondary to the coding step**.

The design of unit tests can be done before coding begins (a preferred agile approach) or after source code has been generated.

Because a module is (usually) not a stand-alone program, **driver** and/or **stub** software must be developed for each unit test.

Unit testing

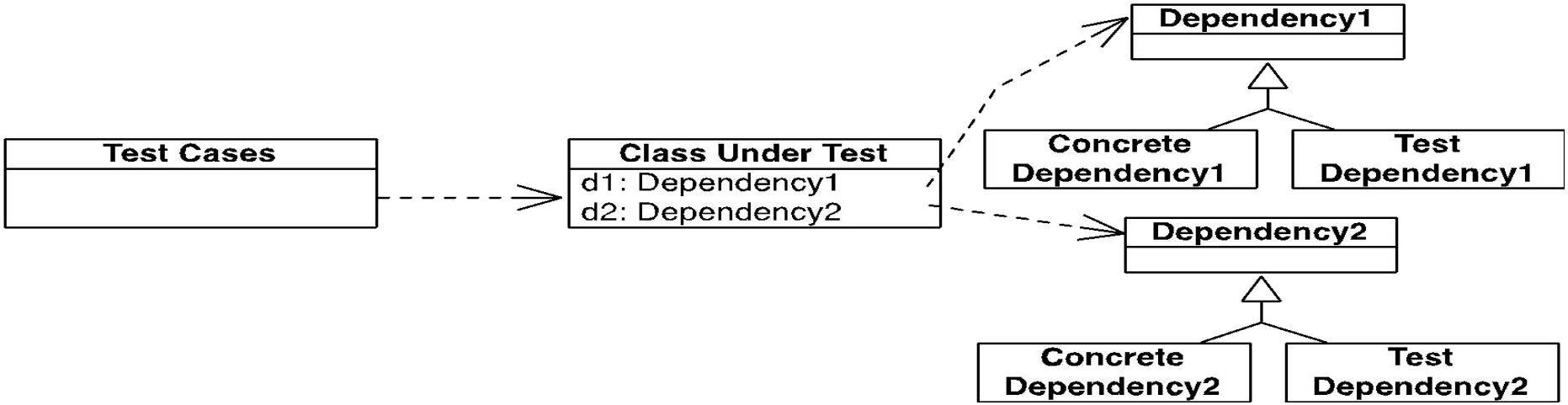
Test Cases has a dependency to the **Class Under Test** in order to be able to test it. The **Class Under Test** also has dependencies that must be included in the test in order for the **Class Under Test** to work.



Would there be any problems with this?

Unit testing

Class Under Test has a dependency to **Dependency1** and **Dependency2**, which are interfaces. The real code for the program is represented by **Concrete Dependency1** and **2**, while test code is stubbed into **Test Dependency1** and **2**.



Unit testing

Driver: In most applications, it is nothing more than a “main program” that accepts test case data, passes such data into the module to be tested, and prints the relevant results.

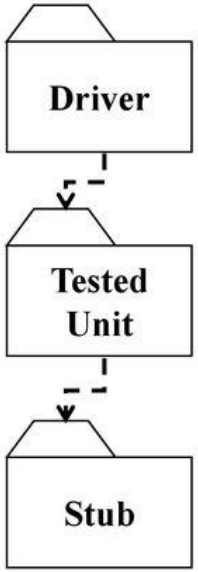
Stub: Dummy modules used to replace a module that is subordinate to (called by or used by) the module to be tested. A stub uses the subordinate modules interface, may do minimal data manipulation, and provides verification of entry and returns.

Drivers and stubs represent overhead. If the drivers and stubs are kept simple, actual overhead is relatively low.

Unit testing is simplified when a component has high cohesion.

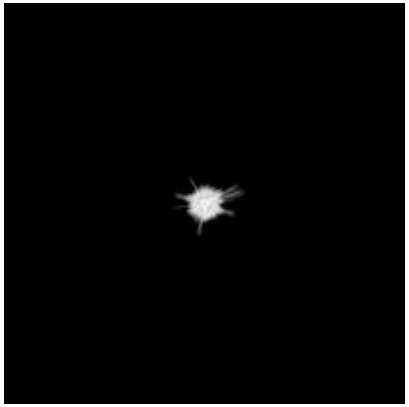
Stubs and drivers

- Driver:
  - A component, that calls the `TestedUnit`
  - Controls the test cases
- Stub:
  - A component, the `TestedUnit` depends on
  - Partial implementation
  - Returns fake values.



Integration testing

Integration testing is a **systematic technique** for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.



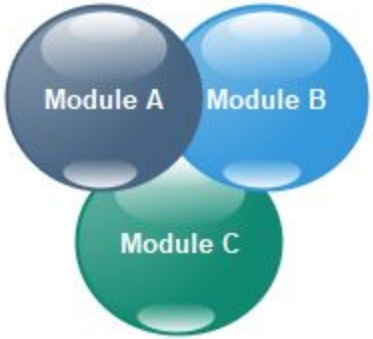
Integration testing strategies:

- Non-incremental (big bang) integration
- Incremental Integration

Non-incremental Integration:

- All modules are combined and tested as a whole
- **Chaos usually results** as a set of errors are encountered.
- Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program.
- Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

Tested in Unit Testing

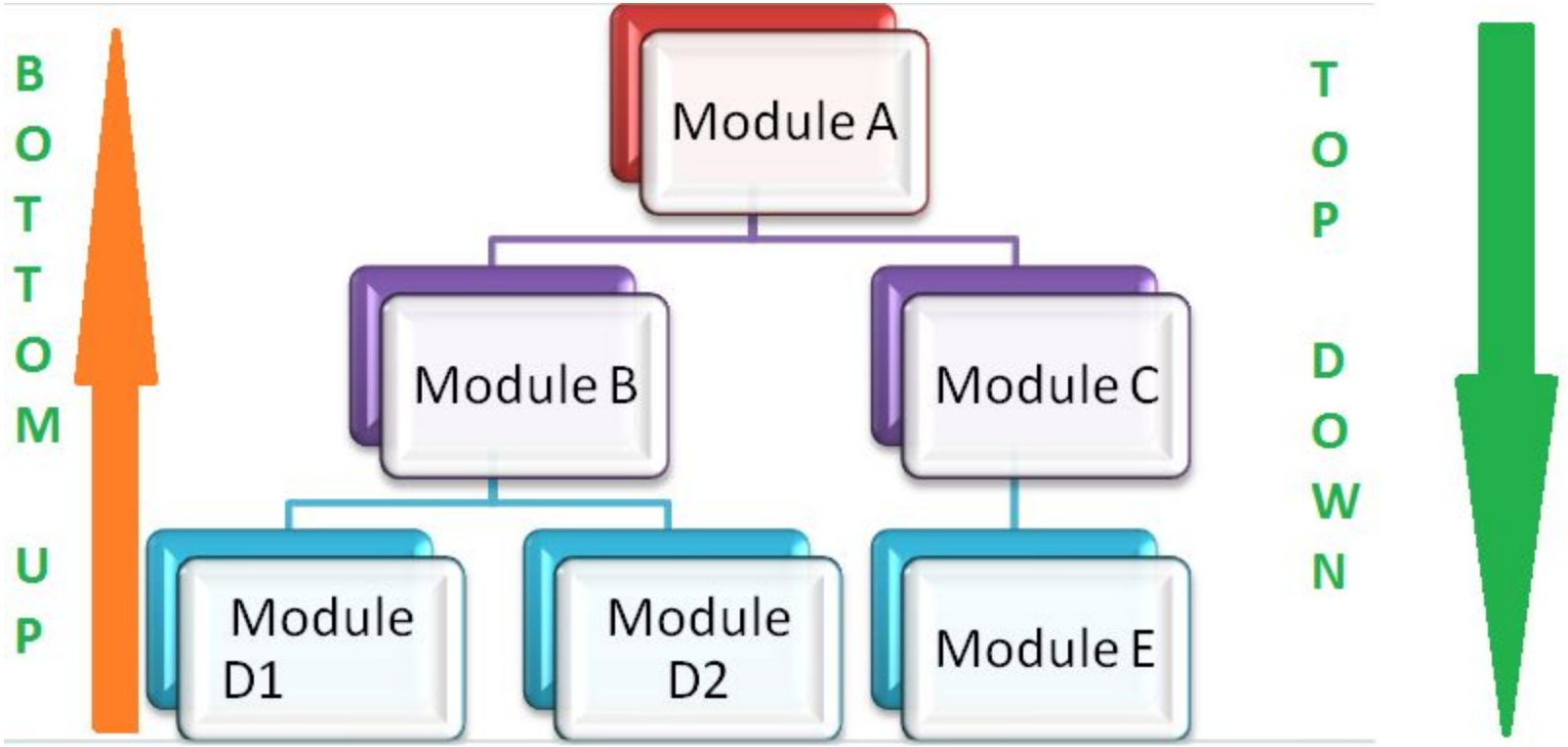


Under Integration Testing

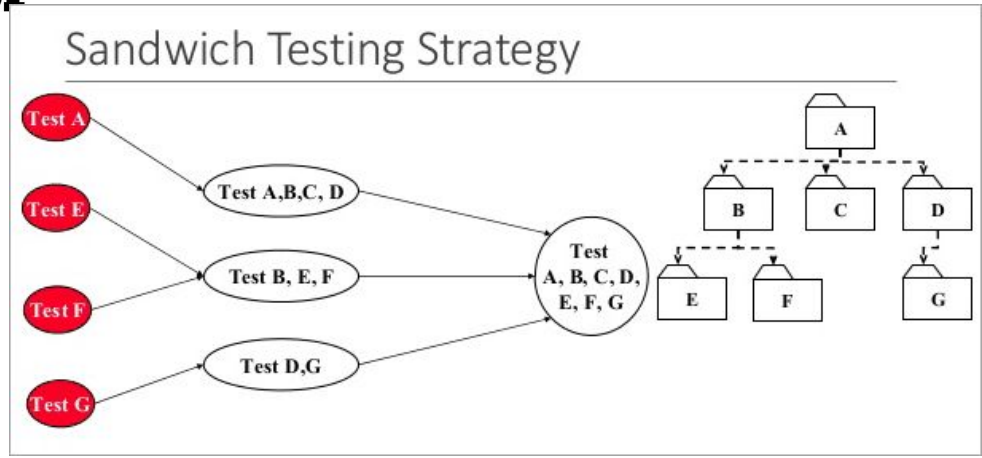
Integration testing

Incremental Integration:

- Program is constructed and tested in small increments.
- Errors are easier to isolate and correct.
- Interfaces are more likely to be tested completely and repeatedly.
- Requires additional overhead in the form of drivers and/or stubs.
- Several approaches to incremental integration:
  - Top-down integration
  - Bottom-up integration
  - Sandwich integration


BOTTOM UP
Module A
Module B
Module C
Module D1
Module D2
Module E
TOP DOWN

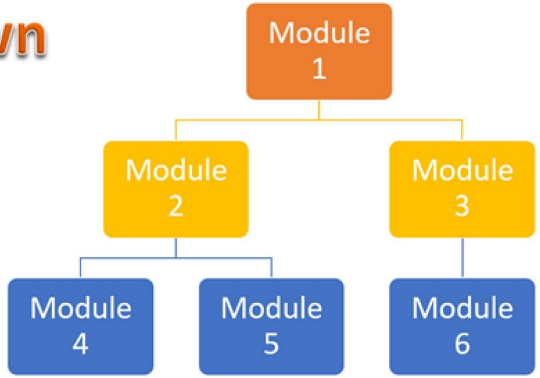
Detailing on Hybrid Integration or Sandwich Testing



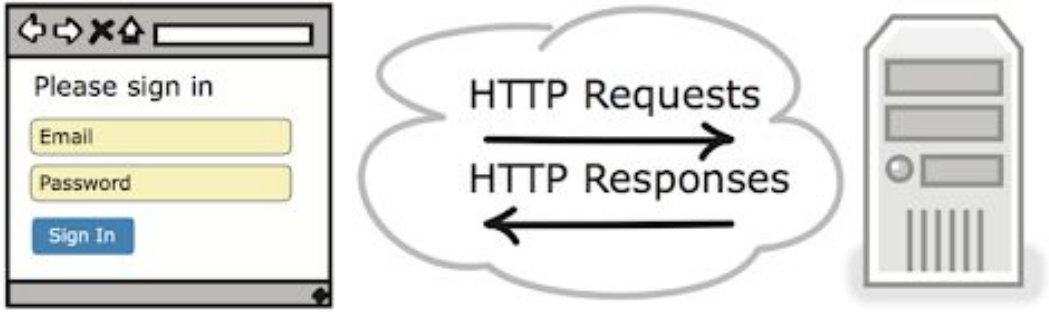
Top Down

Hybrid

Bottom Up



TestOrigen



Is this a possible integration test?

```
test "login with valid credentials" do
  get login_path
  post login_path, session: { email: 'user@test.com', password: 'password' }
  follow_redirect!
  assert_select "h1", "Welcome"
end
```

Validation testing

Validation (**acceptance**) testing is achieved through a series of tests that demonstrate conformity with requirements.

- The focus is at the requirements level
  - On things that will be immediately apparent to the end-user.

- Most software product developers use a process called **alpha** and **beta** testing to uncover errors that only the end-user seems able to find.

Validation testing

Alpha Testing:

- Conducted at the developer's site by end-users.
- Software is used with the developer "looking over the shoulder" of typical users and recording errors and usage problems.
- Conducted in a controlled environment.

Beta Testing:

- Conducted at one or more end-user sites.
- Developer is generally not present.
- "Live" application of software in an environment that cannot be controlled by the developer.
- End-user records all problems (real or imagined) that are encountered during beta testing and reports them to the developer.

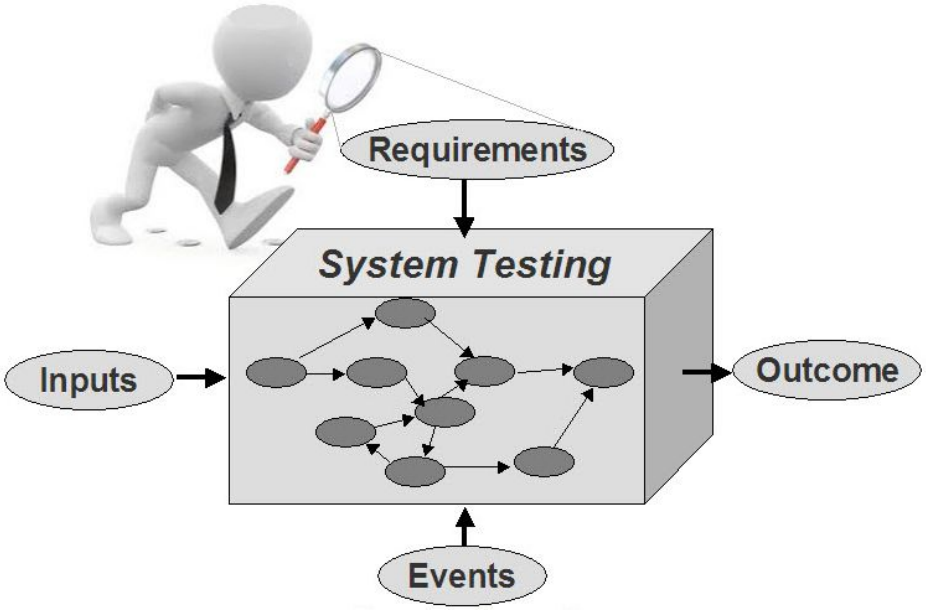
Alpha Test	Beta Test
Performed by developers	Performed by Customers
It is conducted for software application	It is conducted for product
Performed in Virtual Environment	Performed in Real Environment
Involve both black and white box testing	Involve both black box testing only

???

System testing

Software is only one element of a larger computer-based system.

System testing is a series of tests whose primary purpose is to fully exercise the computer-based system.



System testing

Types of system testing:

- **Volume**: Test with large volumes of data
- **Stress**: Test with heavy loads or stresses
- **Usability**: Test human-factor or usability problems
- **Security**: Attempt to subvert security checks
- **Performance**: Attempt to show performance objectives are not met
- **Storage**: Attempt to show storage objectives are not met
- **Configuration**: Test the various configurations of the program
- **Installability**: Install procedures are tested
- **Procedure**: Test any procedures involving people

- **Compatibility**, **Reliability**, **Recover**, **Serviceability**, and **Documentation** Testing

Test Case Name	Test Addition
Test Case ID	1
Test Case Description	Unit test for `addNumbers`(int a, int b) function
Test Type	[Unit, Integration, **System**]
Inputs	[a=0, b=1]
Expected Output	1
Actual Output	-5
Pass/Fail?	FAIL
Requirement Link(s)	FR1

What kinds of tests could you do for your term project?

?!?!??

??

!?!??!?!??!