Software Quality Engineering

05. Integration Testing

Achiya Elyasaf

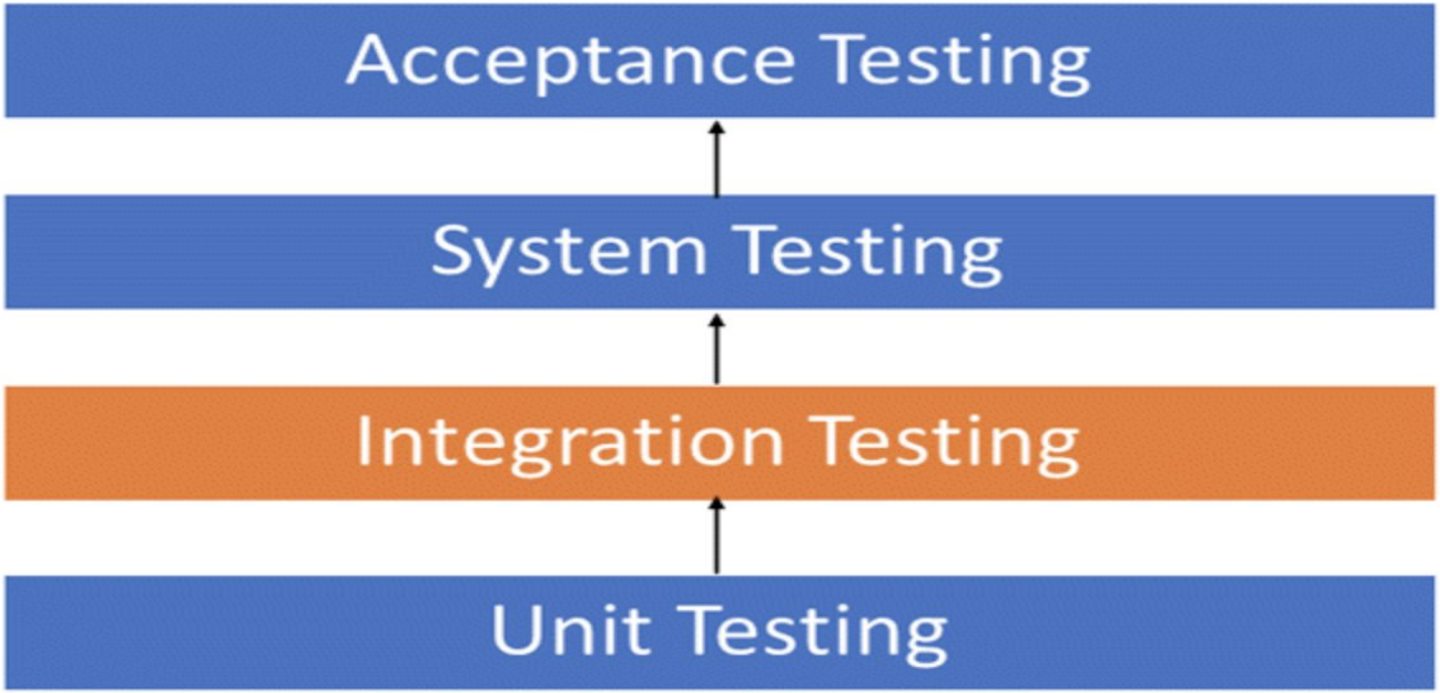
אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

הנדסת מערכות תוכנה ומידע
Software and Information Systems Engineering

Software Testing and QA Theory and Practice, Naik andTripathy

Today’s Agenda

- Introduction
- Integration Techniques
- Software & Hardware Integration
- Integration-Tests Planning


Acceptance Testing
System Testing
Integration Testing
Unit Testing

The Concept of Integration Testing

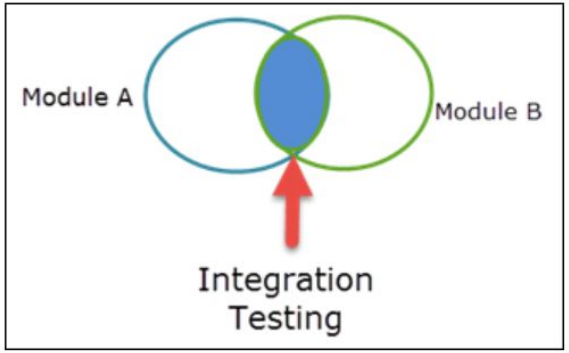
\ Unit testing ✓

\ Test modules together

\ Build a “working” version of the system

\ complete when the system is fully integrated

- \ All the test cases have been executed
- \ All the severe and moderated defects found have been fixed



The Concept of Integration Testing

\ A systematic technique for constructing the program structure

\ Conducting tests to uncover errors associated with interfacing

\ Integrate entire program?

    \ Correction of errors is difficult

    \ One error leads to another

\ □ Incremental integration

Integration Testing - Advantages

\ Defects are detected early

\ It is easier to fix defects detected earlier

\ Earlier feedback on the health and acceptability of the **individual modules** and on the **overall system**

\ Scheduling of defect fixes is flexible, and it can overlap with development

\ Correctness of next steps

Interfaces

\ Interfaces - access services provided by other modules

\ Passing control and data between modules

Interface errors: associated with structures existing outside the local environment of a module, but which the module uses

Types of Interfaces

\Procedure call interface

\Shared memory interface

\Message passing interface — send data to another module (common in client/server)

Different Types of Interface Errors

\ Construction

\ Inadequate functionality

\ Location of functionality

\ Changed Functionality

\ Added functionality

\ Misuse of interface

\ Misunderstanding of interface

\ Data structure alteration

\ Inadequate error processing

\ Additions to error processing

\ Inadequate post-processing

\ Inadequate interface support

\ Initialization/value errors

\ Violation of data constraints

\ Timing/performance problems

\ Coordination changes

\ Hardware/software interfaces

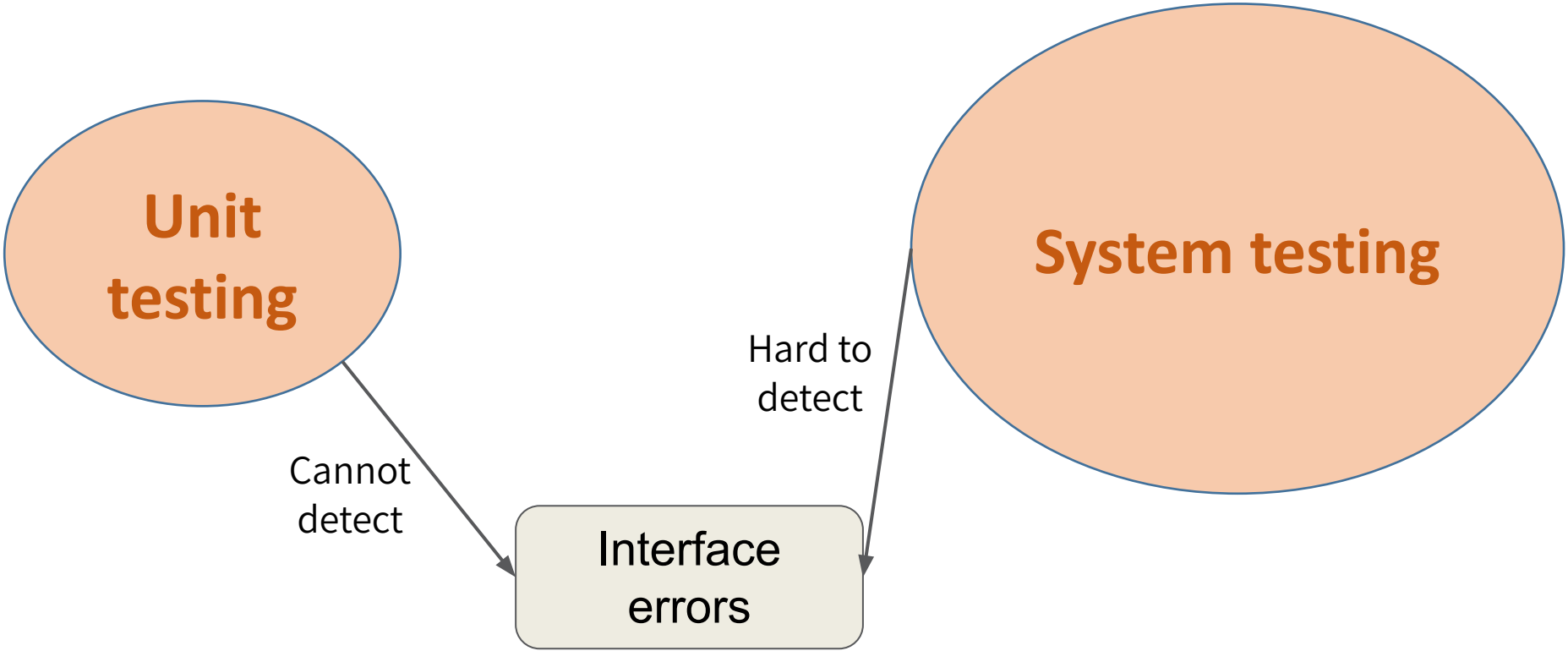
Different Types of Interface Errors

- Construction — In C: the header & implementation don't match
- Inadequate functionality — Interface not doing what it should do
- Location of functionality — Functionality is implemented somewhere else
- Functionality changed — changing one module without adjusting others
- Added functionality — adding functionality to a module after it was tested without a formal change request
- Misuse of interface — error in using interface: e.g., wrong parameter order
- Misunderstanding of interface — e.g., the passed array should be sorted
- Data structure alteration — e.g., the passed array assumed to be with max length of 1000

Different Types of Interface Errors

- Inadequate error processing
  - Calle fails to handle returned error
- Additions to error processing
  - Called module change error message but calle did not change error handling
- Inadequate post-processing
  - Eg., resources not released
- Inadequate interface support
  - Eg., temperature in celsius instead of Fahrenheit
- Initialization/value errors
  - Added functional
  - Eg., pointer changes
- Violation of data constraints
  - Eg., first param is smaller than second
- Timing/performance problems
  - Misunderstanding
  - e.g.: race condition
- Coordination changes
  - A module coordinates two other modules fails
- Hardware/software interfaces
  - Software design to wrong hardware version

Why integration testing?



Granularity of System Integration Testing

\Intra-system testing

\ Low-level: combining the modules together to build a cohesive system

\Inter-system testing

\ High-level: interfacing independently tested systems

\ Pairwise testing

\ Between intra- and inter-system: integrate two at a time, assume the other system is okay

System Integration Techniques

- Incremental
- Top-down
  - Stubs are used if modules aren’t ready
- Bottom-up
- Sandwich (hybrid)
  - UI with stubs, functions with drivers, target (middle) layer
- Big-bang – all connected
  - Hard to isolate errors

Incremental

\A series of test cycles

\In each test cycle, a few more modules are integrated

\Each cycle should be self-contained:

contains all necessary code to support a certain functionality, and **stable**

\The complete system is built, cycle by cycle

Incremental

Depends on:

\ Number of modules in the system

\ Relative complexity of the module

\ Relative complexity of the interfaces between the modules

\ Number of modules needed to be clustered together in each test cycle

\ Whether the modules to be integrated have been adequately tested before

\ Turnaround time for each test-debug-fix cycle

How many cycles?

Incremental

\ Build process:

- \ Build and test each module
- \ Link modules together
- \ Verify linkage (integration tests)

\Final build: a candidate for system testing

\Large numbers of modules and/or complicated interfaces □ complex build process

Incremental

complex builds —> use a CI (Continuous Integration) tool

- faster delivery of the system
- small incremental testing from build to build

Incremental

A release note containing the following information accompanies a build:

\ What has changed since the last build?

\ What outstanding defects have been fixed?

\ What are the outstanding defects in the build?

\ What new modules, or features, have been added?

\ What existing modules, or features, have been enhanced, modified, or deleted?

Incremental

Test strategy for each build addresses:

\ What test cases need to be selected from the System Integration Testing (SIT) test plan?

\ What previously failed test cases should now be re-executed in order to test the fixes in the new build?

\ How to determine the scope of a partial regression tests?

\ What are the estimated time, resource demand, and cost to test this build?

Incremental – Daily Build

\ Facilitates to a faster delivery of the system

\ Puts emphasis on small incremental testing

\ Steadily increased number of test cases

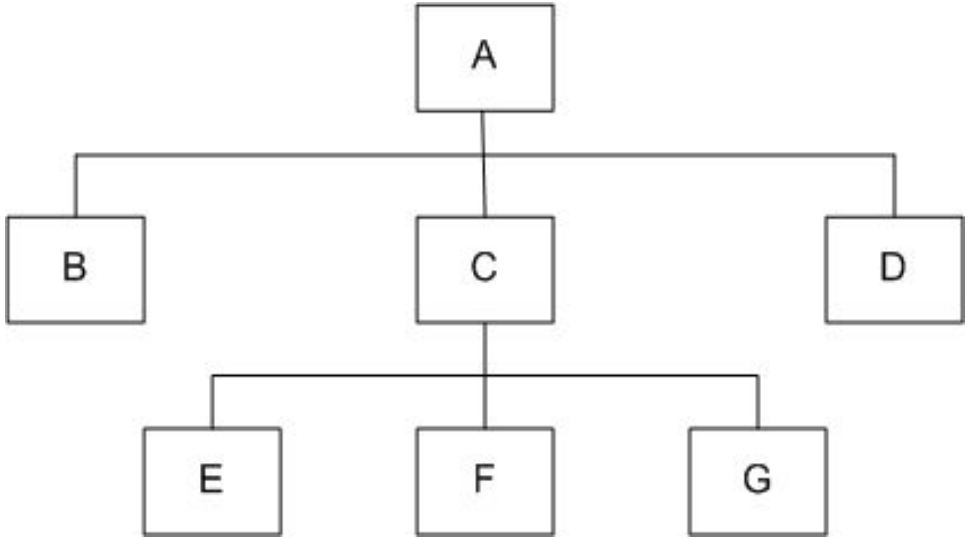
\ The system is tested using automated, re-usable test cases

\ An effort is made to fix the defects that were found within 24 hours

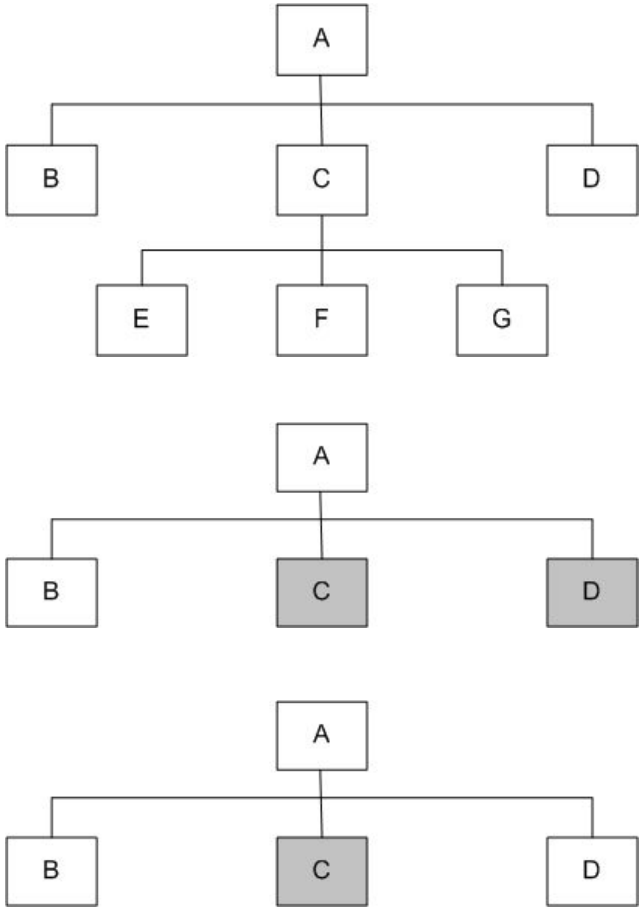
\ Prior version of the build are retained for references and rollback

\ A typical practice is to retain the past 7-10 builds

A Modular System – Top Down



Top-down

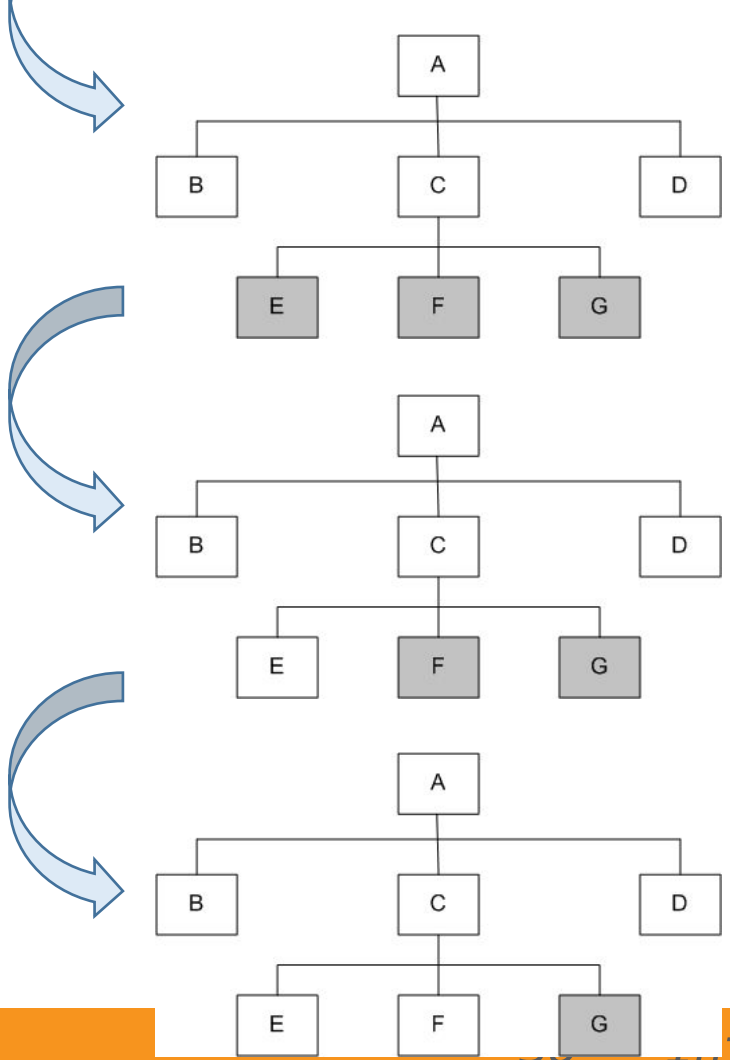


Terminals: B,D,E,F,G

Test A+B **integration** using stubs C` and D`

Replace D` with D and:
— Test A+D **integration**
— **Regression test** of A+B

Top-down



Replace C` with C,
stub E, F, G, and:
— Test A+C **integration**
— **Regression test** of A+B+D

Top-down

Advantages

\ Easier Isolation of interface errors — check if the interface error is related to the newly integrated module or to the previously integrated modules

\ Test cases for integration of module are reused in regressions

\ Inputs to test cases are applied to the top-level module, so these test cases can be designed for reusing in system-level tests

Top-down

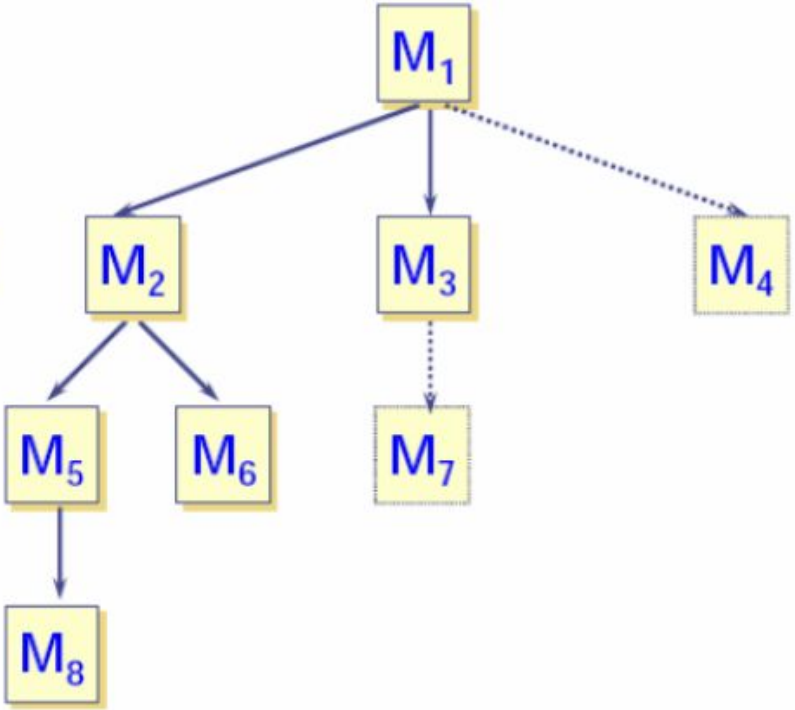
Limitations

\ Low-level modules might contain meaningful functionality but initially they are replaced by stubs. It should be considered while designing the test cases.

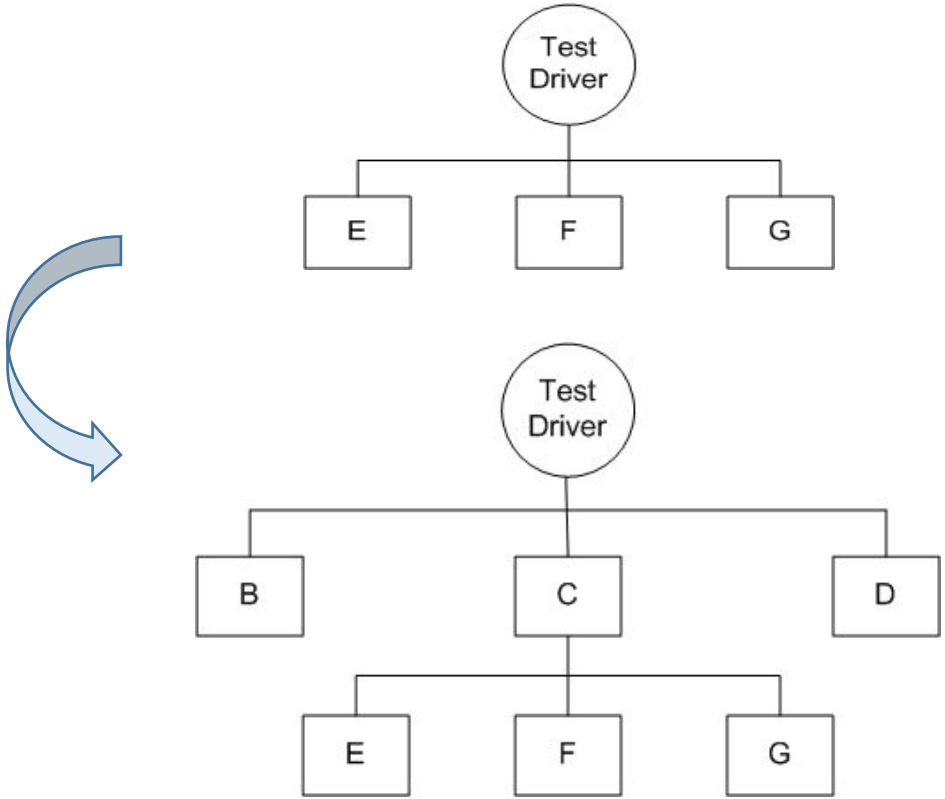
\ Hard to think of input vector that will test integration of low-level modules.

\ The more levels we have, test case selection and **stubs design** becomes more difficult because of the limited behavior of stubs.

- **Top-Down Testing with Depth-First**
  - M1, M2, M5, M8
  - M6
  - M3, M7
  - M4
- **Top-Down Testing with Breath-First**
  - M1
  - M2, M3, M4
  - M5, M6, M7
  - M8



Bottom-up



Integrate lowest level using test (mimics C)

Add middle layer to integration (mimic A)

Bottom-up

Advantages

\ Simple to design test driver (a simplification of the actual module's behavior)

\ If low-level modules and their combined functions are often invoked by other modules, then test them first for meaningful & effective integration with other modules

Bottom-up

Disadvantages

\ Major faults are detected towards the end of the integration process,

because major design decision are embodied in the top-level modules

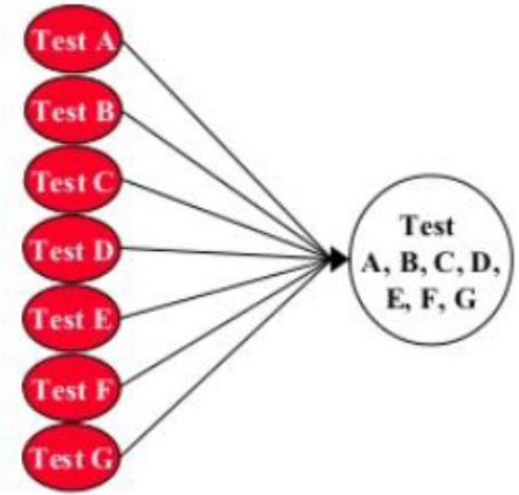
\ Test engineers can not observe system-level functions from a partly integrated system.

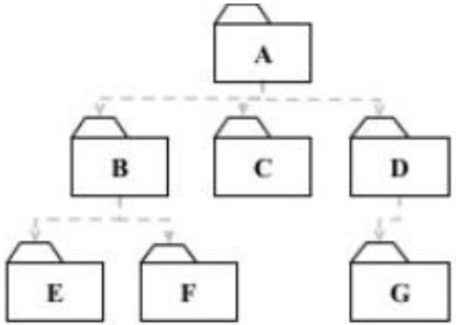
In fact, they can not observe system-level functions until the top-level test driver is in place

Big-bang approach

\ First all the modules are individually tested

\ Next all those modules are put together to construct the entire system which is tested as a whole





Big-bang approach

Limitations:

\ Difficult to know the root cause of the failure

\ Higher chances for critical bugs in production

\ Time consuming

Why using this approach?

\ No good reason…

\ Generally executed by developers who follows the ‘Run it and see’ approach (anyone said students?)

Sandwich Approach

\ A system is integrated using a mix of top-down, bottom-up, and big-bang

\ A hierarchical system is viewed as consisting of three layers

\ The bottom-up approach is applied to integrate the modules in the bottom-layer

\ The top layer modules are integrated by using top-down approach

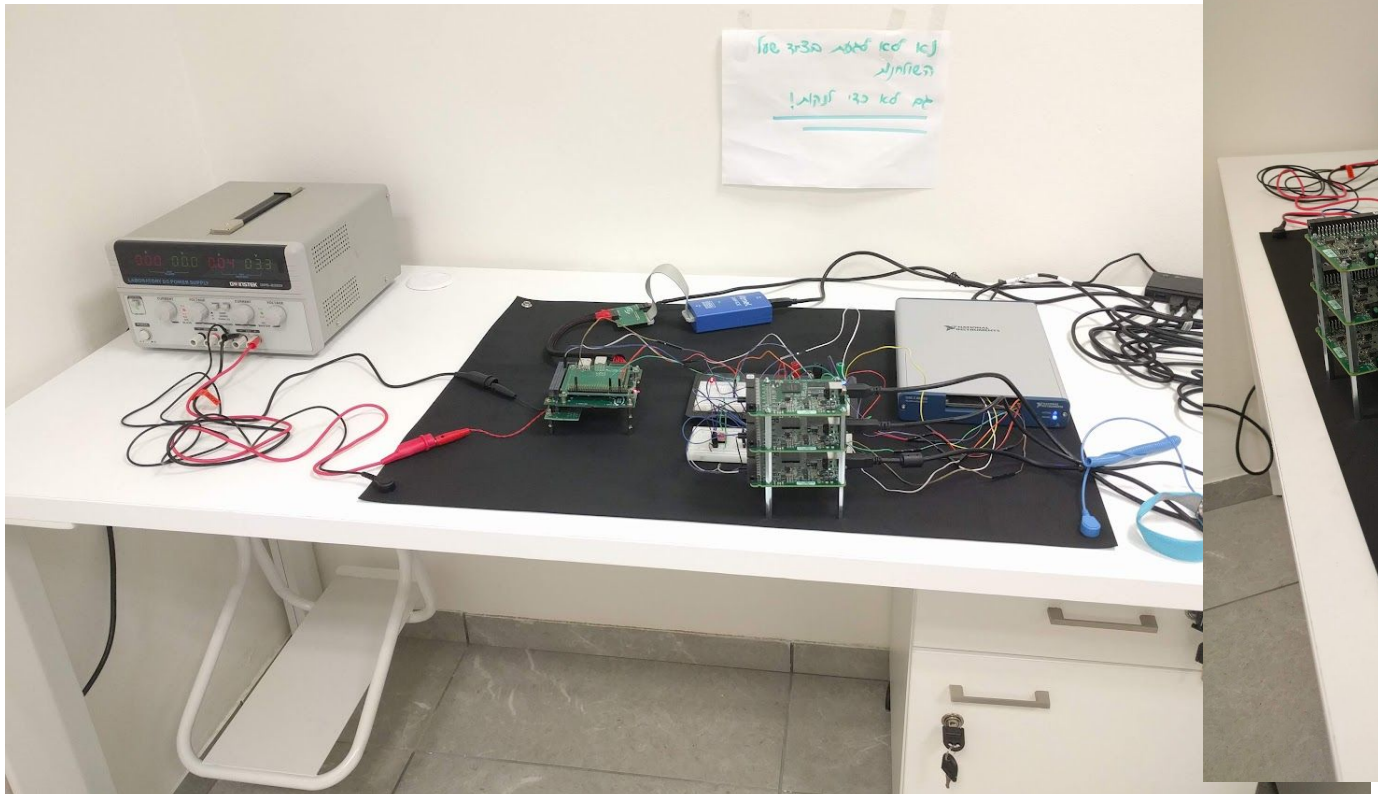
\ The middle layer is integrated by using the big-bang approach after the top and the bottom layers have been integrated

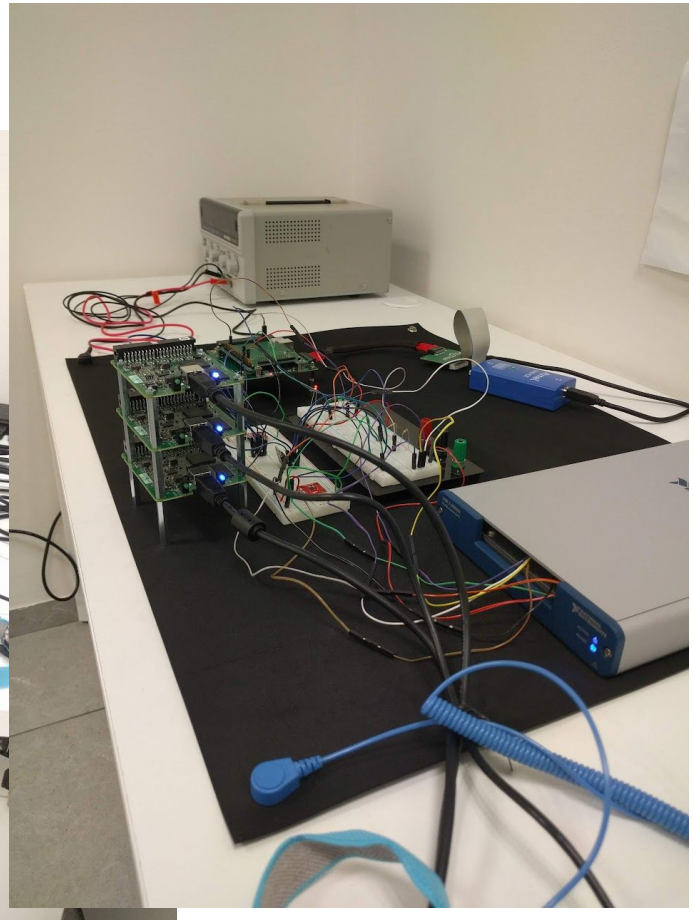
Software and Hardware Integration

Challenging:

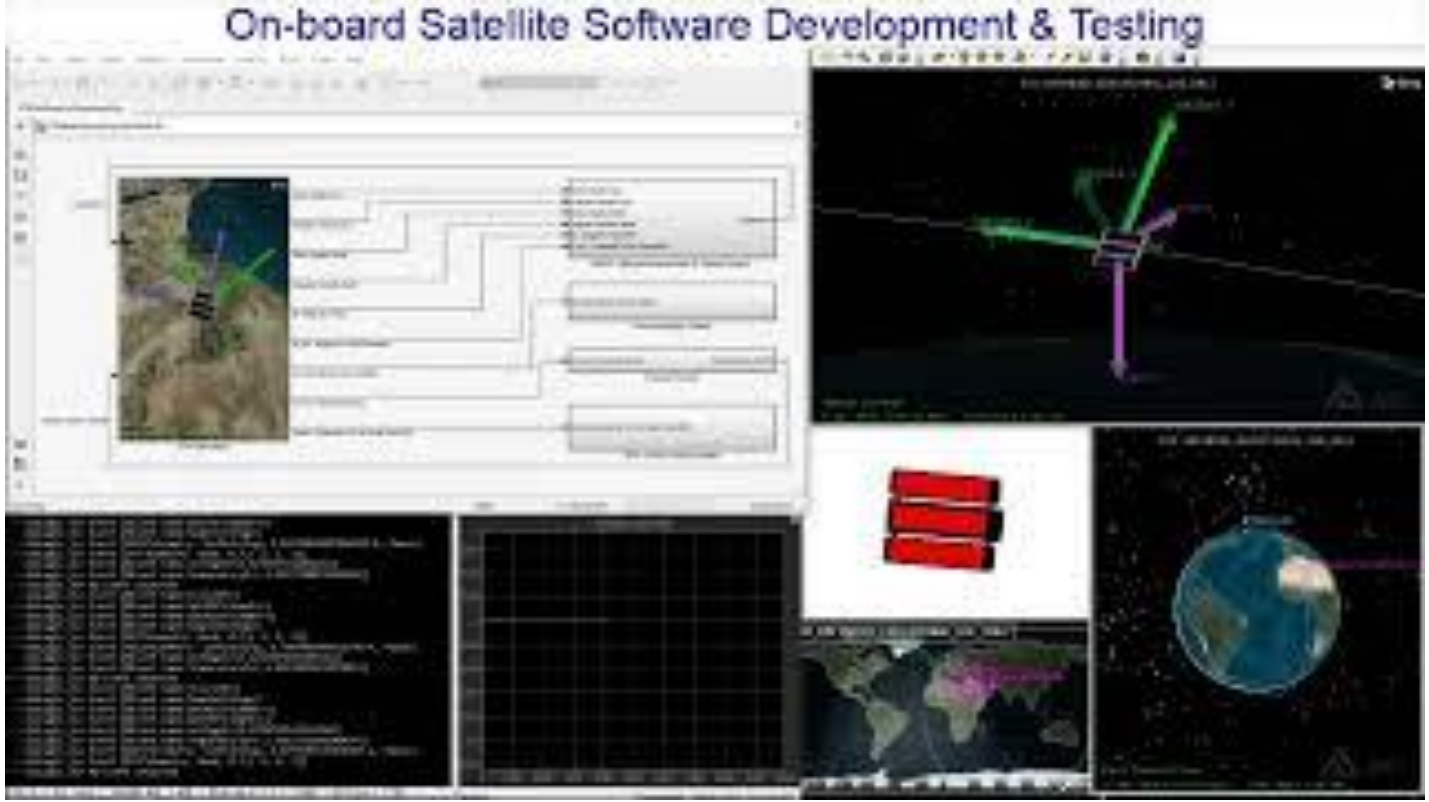
- Hardware & software are developed in parallel. We can’t integrate them at the end
- Software may act different in reality
- Hardware may act differently in various environments (e.g., space, ground, while speeding, etc.)

Hybrid Laboratory





Hybrid Laboratory



Software and Hardware Integration

\ Integration is often done in an iterative manner

\ A software image with a minimal number of core modules is loaded on a prototype hardware

\ A small number of tests are performed to ensure that all the desired software modules are present in the build

\ Next, additional tests are run to verify the essential functionalities

\ The process of assembling the build, loading on the target hardware, and testing the build continues until the entire product has been integrated

Hardware Design Verification Tests

Hardware-engineering process:

\ Planning and specification

\ Design, prototype implementation, and testing

\ Integration with the software system (previous slide)

\ Manufacturing, distribution and field service

❖ A hardware **Design Verification Test** (DVT) plan is prepared and executed by the hardware group before the integration with software system

Hardware Design Verification Tests

\ Diagnostic Test

\ Electrostatic Discharge Test

\ Electromagnetic Emission Test

\ Electrical Test

\ Thermal Test

\ Environment Test

\ Equipment Handling and Packaging Test

\ Acoustic Test

\ Safety Test

\ Reliability Test

Hardware and Software Compatibility Matrix

\ HW and SW compatibility matrix

\ Different revisions for HW and SW

\ Engineering Change Order (ECO) — formal document describe changes to hardware & software

- \ Includes the hardware-software compatible matrix
- \ Distributed to the operation, customer support and sales teams of the organization

Citrix Hardware platforms/Software releases	10.5	11	11.1	12	12.1	13.0
MPX 5500	10.5–50.10	11.0–62.10	11.1–47.14	12.0–41.16	12.1–48.13	13.0–36.27
MPX 5550/5650	10.5–50.10	11.0–62.10	11.1–47.14	12.0–41.16	12.1–48.13	13.0–36.27
MPX 7500	10.5–50.10	11.0–62.10	11.1–47.14	12.0–41.16	12.1–48.13	13.0–36.27
MPX 8900 FIPS certified (In Progress)	X	X	X	X	12.1–55.190	X
MPX 14040-40S/14060-40S/14080-40S/14100-40S	10.5–59.71	11.0–62.10	11.1–47.14	12.0–41.16	12.1–48.13	13.0–36.27
MPX 14030 FIPS/14060 FIPS/14080 FIPS	X	X	11.1–51.21	12.0–41.16	12.1–48.13	13.0–36.27
MPX 15020/15030/15040/15060/15080/15100	X	X	11.1–60.13	12.0–60.10	12.1–50.31	13.0–36.27
MPX 15000-50G	X	X	11.1–56.15	12.0–57.24	12.1–50.31	13.0–36.27
MPX 15000-50G FIPS certified (In Progress) (See the note after the table)	X	X	X	X	12.1–55.190	X
MPX 22040/22060/22080/22100/22120	10.5–50.10	11.0–62.10	11.1–47.14	12.0–41.16	12.1–48.13	13.0–36.27
MPX 24100/24150	10.5–51.10	11.0–62.10	11.1–47.14	12.0–41.16	12.1–48.13	13.0–36.27

Test Plan

1. Scope of Testing
2. Structure of the Integration Levels a. Integration test phases b. Modules or subsystems to be integrated in each phase c. Building process and schedule in each phase d. Environment to be set up and resources required in each phase
3. Criteria for Each Integration Test Phase n a. Entry criteria b. Exit criteria c. Integration Techniques to be used d. Test configuration set-up
4. Test Specification for Each Integration Test Phase a. Test case id# b. Input data c. Initial condition d. Expected results e. Test procedure How to execute this test? How to capture and interpret the results?
5. Actual Test Results for Each Integration Test Phase
6. References
7. Appendix

Table 7.3: A framework for System Integration Test (SIT) Plan.

Test Plan for System Integration

Entry Criteria
Software functional and design specifications must be written, reviewed and approved.
Code reviewed and approved.
Unit test plan for each module is written, reviewed, and executed.
100% unit tests passed.
All the check-in request form must be completed, submitted, and approved.
Hardware design specification written, reviewed, and approved.
Hardware design verification test written, reviewed, and executed.
100% design verification tests passed.
Hardware/software integration test plan written, reviewed, and executed.
100% hardware/software integration tests passed.

Test Plan for System Integration: Inputs

\ Software Requirements Data

\ Software Design Document

\ Software Verification Plan

\ Software Integration Documents

Test Plan for System Integration: Activities

\ Based on the High and Low-level requirements create test cases and procedures

\ Combine low-level (modules) builds that implement a common functionality

\ Develop a test harness (test automation framework)

\ Test the build

\ Once the test is passed, the build is combined with other builds and tested until the system is integrated as a whole.

\ Re-execute all the tests on the target processor-based platform, and obtain the results

Test Plan for System Integration

\ Interface integrity

\ Functional validity

\ End-to-end validity

\ Pairwise validity

\ Interface stress

\ System endurance

Test Plan for System Integration

Exit Criteria
All code completed and frozen and no more modules to be integrated.
100% system integration tests passed.
No major defect is outstanding.
All the moderate defects found in SIT phase have been fixed and re-tested.
Not more than 25 minor defects are outstanding.
Two weeks system uptime in system integration test environment without any anomalies, i.e., crashes.
System integration tests results are documented.

Test Plan for System Integration: Outputs

\ Integration test reports

\ Software Test Cases and Procedures [SVCP].

Off-the-shelf Component Integration

\ **Wrapper:** a component built to isolate the underlying components from other components

\ **Glue:** a component for combining different components

\ **Tailoring:** the ability to enhance the functionality of a component:

- \ Achieved by adding elements to a component to enrich it with a functionality not provided by vendor
- \ Does not involve modifying the source code of the component (e.g., plugins)

Off-the-shelf Component Testing

Tests for a COTS (commercial off-the-shelf) component:

\ **Black-box component testing:** determine the quality of the component

\ **System-level fault injection testing:** determine how well a system will tolerate a failing component

\ **Operational system testing:** determine the tolerance of a software system when the COTS component is functioning correctly

J B Rainsberger

INTEGRATED TESTS ARE A SCAM

https://www.youtube.com/watch?v=VDfX44fZoMc
http://blog.thecodewhisperer.com/series#integrated-tests-are-a-scam