ECEN 478: Senior Design

ECEN 478

Dr. Mahmoud Nabil Mahmoud
mnmahmoud@ncat.edu

North Carolina A & T State University

April 5, 2022

Outline

Motivation

- Development is accompanied by "bugs."
- Catching bugs early saves money
  - The further a bug progresses the more impact it has on the system
  - A bug fix requires all related modules to be retested.
  - A bug fix may require redesigning related modules.
- For example
  - PCB design flaw
  - VLSI layout error
  - Subtle coding flaw
- Testing doesn't remove bugs, it just makes it less likely they exist.

Learning Objectives

By the end of this chapter, you should:

- Understand the concepts of black box tests, white box tests, observability, and controllability.
- Understand the principles of debugging.
- Understand when a unit test is used and how it is constructed.
- Understand when an integration test is used and how it is constructed.
- Understand when an acceptance test is used and how it is constructed.

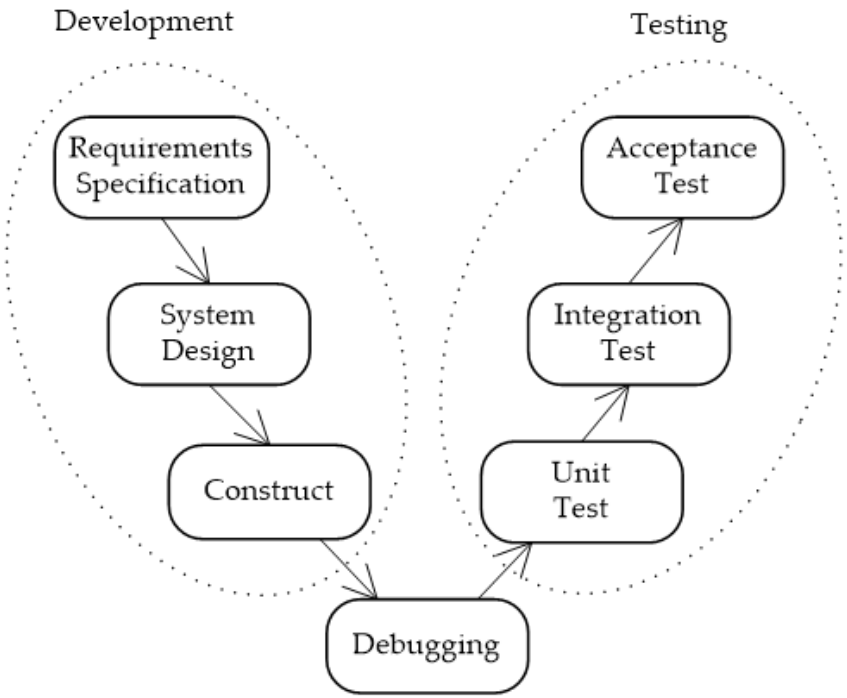
What Does "Testing" Mean in Engineering Design?

- Ensuring that a design module gives the correct output combination, for each given input combination.
- Levels of Testing
  - Unit testing – Testing of the implementation of individual terminal modules in a block diagram
  - Integration testing - Testing of the combinations of multiple module implementations (non-terminal modules, including Level-0)
- All tests should be based on some part of the **Requirements Specification**

Outline

Testing Principles

- Testing proceeds with design process
  - Write tests while designing modules,
  - Perform tests while implementing modules
- The Test-Vee illustrates this process



Testing

- Quality design documents
  - Requires you to reason about system behavior
  - Gives you clear design goals
  - Look at your system from the tester's perspective
- How should we perform testing?
  - Tradeoff
    - Apply every possible input, or
    - Apply selected inputs which might yield errors
- Test should be chosen to increase likelihood of finding an error.

Why Test Cases?

- Test cases define exactly what the module must do.
- Testing prevents feature creep, since the development of a module is complete when its test is passed.
- Test cases motivate developers by providing immediate feedback.
- Test cases force designers to think about extreme cases.
- Test cases are a form of documentation.
- Test cases force the designer to consider the design of the module before building it.

Types of test

- Black box
  - No knowledge of internal organization
  - Only access input and outputs
  - Change inputs and observe outputs
- White box
  - Knowledge of internal organization
  - Might have expectation of fault model
  - Create test instance which reveal physical or logical errors

Testable

- Testability – failure of a component can be
  - Quickly detected
  - Quickly located
- Controllability
  - When any node of the system can be set to a desired value
  - Black box has no controllability
- Observability
  - When any node of the system can be measured.
  - Black box has low observability

Stub

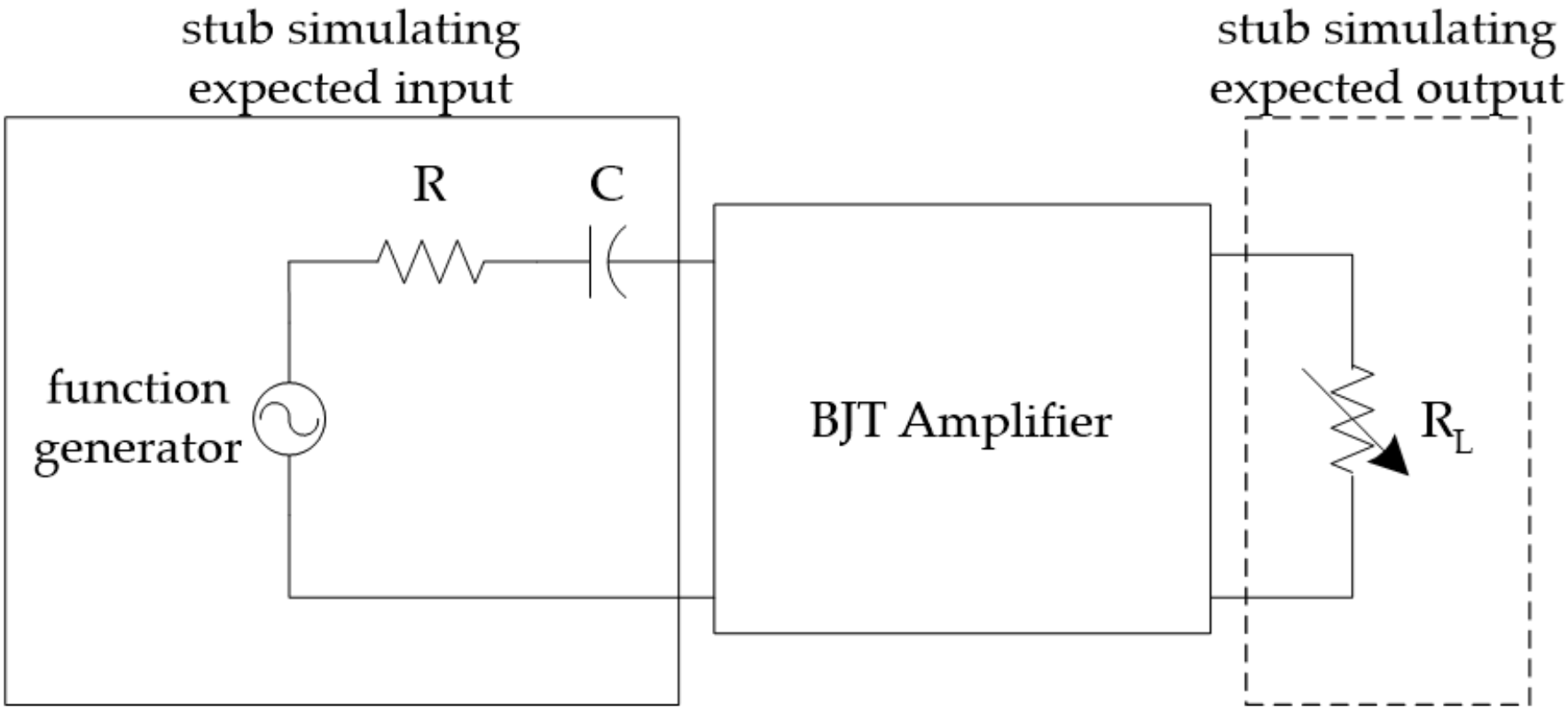
Stub

- A placeholder for future functionality
- A device which mimics subsystem
  - Simulates input (and/or outputs), or monitors output
  - For a unit under test (UUT)
  - Insure good behavior
    - before wreaking havoc on the rest of the system
    - While waiting for implementation of other modules (units)

For example

- A function generator for a audio input
- A printf() instead of a file write
- DIP switch instead of a bus connection

Stub example



Outline

Test case properties

- **Accurate** - The test should check what it is supposed to and exercise an area of intent.
- **Economical** - The test should be performed in a minimal number of steps.
- **Limited in complexity** - Tests should consist of a moderate number (10-15) of steps.
- **Repeatable** - The test should be able to be performed and repeated by another person.
- **Appropriate** - The complexity of the test should be such that it is able to be performed by other individuals who are assigned the testing task.
- **Traceable** - The test should verify a specific requirement.
- **Self cleaning** - The system should return to the pre-test state after the test is complete.

Constructing Tests

Debugging

- Bohrbugs
  - Bohrbugs are reliable bugs
  - The error is always in the same place
  - An electrons having a definite position
  - Solution = set a good trap
- Heisenbugs
  - Innocuous changes of input yield buggy behavior
  - May not be reproducible
  - They seemingly move around within a system
  - Electron is elusive density function
  - Solution = think outside the box

Debugging process

- Observe the problem under different operating conditions
- Form a hypothesis as to what the potential problem is
- Conduct experiments to confirm or eliminate the hypothesized source of the problem
- Repeat until the problem is eliminated

General Testing Guidelines

- Check easiest problems first
  - You can perform more in a given time
- Start at lowest levels of abstraction
  - Upper levels rely on lower level
- Example
  - Is the system powered up?
  - Is the testing equipment adjusted properly?
  - Are the bus lines being correctly manipulated?
  - Have you initialized the system?
  - Are you printing out the right variable/type?

Unit Testing

- A **unit test** is a test of the functionality of a system module in isolation (terminal module)
- Should be traceable to the detailed design.
- Consists of a set of test cases
- Each test case establishes that a subsystem performs a single unit of functionality to some specification (engineering requirement).
- Test cases should be written with the express intent of uncovering undiscovered defects.

Unit Test – continued

- Write unit test during implementation;
- Why?
  - Facilitates white box test development
  - Thinking about test situations and conditions may lead the designer to uncover errors before they are ever designed into the system.
  - Unit tests need to be written with an understanding of the internal organization of the component, and a system is best understood during its development.

Unit test – example and definitions

```
if (16 < Celsius Temperature < 32)
  Fahrenheit Temperature = ROM[input-16];
else
  Fahrenheit Temperature = (9* Celsius Temperature)/5 + 32;
```

1. Processing path – Sequence of instructions or states form start to end of a computation block. How many paths for this example?
2. Test coverage – Extent to which all possible processing path can be covered by a test
3. Path-complete coverage – 100% test coverage. How can it be achieved for this example?

- What inputs are used to check the ROM?
- What inputs check the conditionals?

Code example

- Determine inputs to check all paths
  - ACTION1
  - ACTION2
  - ACTION3

```
if ((x >= 30) && (x <= 39)) {
    ACTION1
} else if ((x >= 80) && (x <= 96)) {
    ACTION2
} else if ((x >= 40) && (x <= 56)) {
    ACTION 3
}
```

Testing Methods

- Matrix Test -
- Step-by-step Test –
- Automated Scripts Tests –

Note: these can be applied to any level of test: unit, integration, or acceptance

Matrix Test

Test Writer: Sue L. Engineer							
Test Case Name:	ADC function test			**Test ID #:**	ADC-FT-01		
Description:	Verify conversion range and clock frequency. Output goes to 0 in presence of null clock.			**Type:**	☐ white box ☑ black box		
Tester Information							
Name of Tester:				**Date:**			
Hardware Ver:	1.0			**Time:**			
Setup:	Isolate the ADC from the system by removing configuration jumpers.						

Test	**V_T**	**Clock**	**Expected output**		**Pass**	**Fail**	**N/A**	**Comments**
			Decimal	**Hexadecimal**				
1	0.0V	10kHz	0	0x000				
5	2.0V	0Hz	0	0x000				
Overall test result:								

Step-by-Step Test

Test Writer: Sue L. Engineer					
	Test Case Name:	Finite State Machine Path Test #1		**Test ID #:**	FSM-Path-01
	Description:	Simulate insertion of money with a mix of nickels and dimes. Verifies FSM, outputs candy in response to a total deposit of $0.30.		**Type:**	☑ white box ☐ black box
Tester Information					
	Name of Tester:			**Date:**	
	Hardware Ver:	1.0		**Time:**	
	Setup:	Make sure that the system was reset sometime prior and is in state $0.00.			

Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Strobe Nickel	State should go to $0.05				
2	Strobe Dime	State should go to $0.15				
3	Wait	State should remain $0.15				
4	Strobe Nickel	State should go to $0.20				
5	Strobe Dime	State should go to $0.25				
6	Nothing	State should go to $0.00				
	Overall test result:					

Integration Testing

- Write integration test during Level-1 design
  - Helps ensure requirements are being met.
  - Helps to firm-up design
  - Requires the designer think about the expected behavior of the subsystems.
  - Requires designer to think about extreme behaviors of subsystems.

Write the Integration Test

- What are the different paths of execution through the system?
- Are all modules exercised at least once during integration testing?
- Have all the interface signals been tested?
- Have all the interface modes been exercised?
- Does the system process information at the required rate and met timing requirements?

Acceptance Testing

- Demonstrate to customer to show that needs are meet
- Might be formal legal document
- Written along with requirements
- Traceable to engineering requirements
- Identifies
  - Scope – how much of the system is tested?
  - Level – how deep will testing be performed?

Outline

Application: Autonomous Robot

- Autonomous navigating robot
- Engineering requirements
  - *The robot's center must stay within 12 to 18 centimeters of the wall over 90% of the course, while traveling parallel to a wall over a 3 meter course.*
  - *The robot's heading should never deviate no more than 10 degrees from the wall's axis, while traveling parallel to a straight wall over a 3 meter course.*

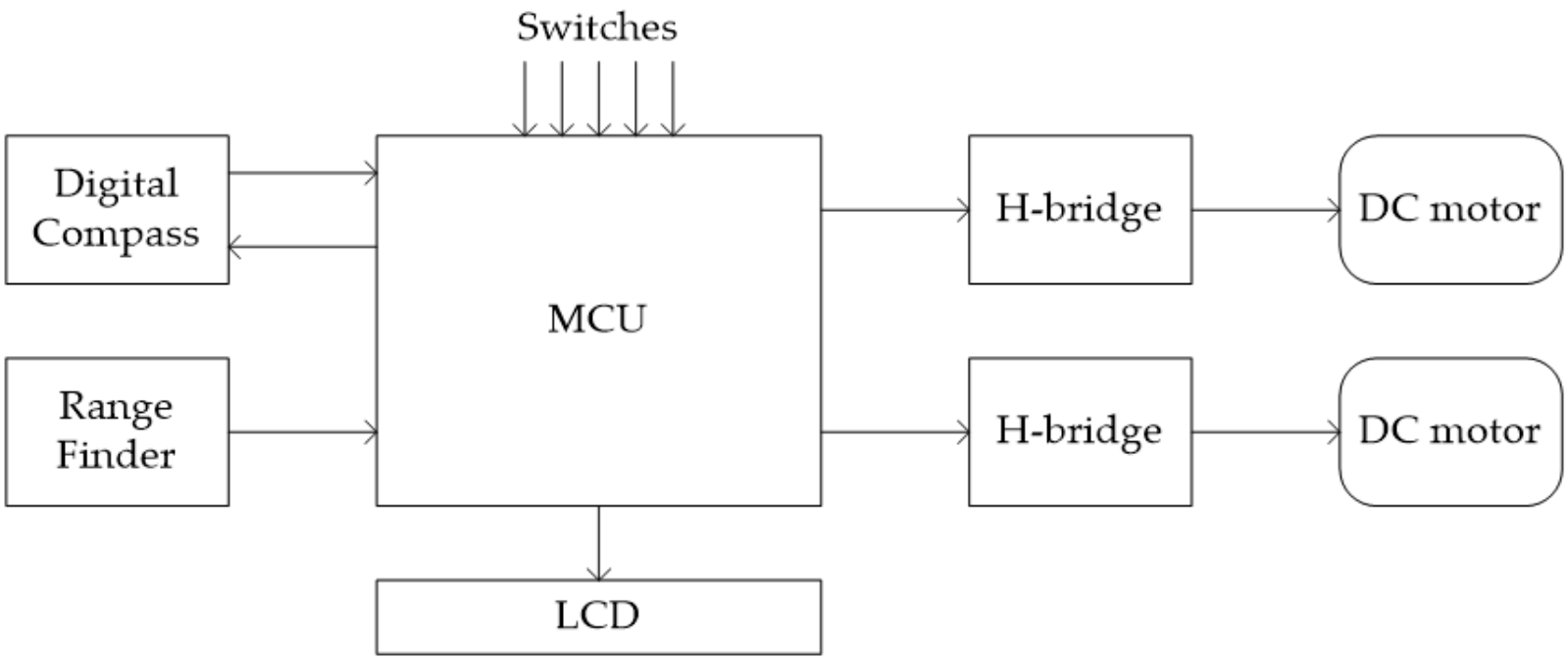
Robot Acceptance Test

Test Writer: Sue L. Engineer				
Test Case Name:	Robot acceptance test #1		**Test ID #:**	Robot-AT-01
Description:	Checks the engineering requirement: *The robot's center must stay within 12 to 18 centimeters of the wall over 90% of the course, while traveling parallel to a wall over a 3 meter course.*		**Type:**	☐ white box ☑ black box
Tester Information				
Name of Tester:			**Date:**	
Hardware Ver:	Robot 1.0		**Time:**	
Setup:	Completed robot should be fully charged and placed on 3 meter test track.			

Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Write a program to monitor the robots position from the wall.	Program should be statically tested to verify accuracy. Should sample wall at a sufficient rate depending on speed.				
2	Put robot on test track, run test, and download data.	The robot should travel down the entire length of the test track and then stop.				
3	Plot test data in a spreadsheet program.	Plot of position vs. time should be within 12 – 18 cm 90% of the time.				
	Overall test result:					

Example: Robot architecture

This is a Bad Level-1 Diagram. Why?
Lets Fix it!



Some Integration Test Possibilities

- MCU + motors + bridge + switches
- Chassis + digital compass + MCU + motors + bridge +LCD
- Chassis + range finder + MCU + motors + bridge

A step-by-step integration test

Test Writer: Sue L. Engineer				
	Test Case Name:	Robot integration test #1	**Test ID #:**	Robot-IT-01
	Description:	Checks interaction of DC motors on the magnetic compass.	**Type:**	☐ white box ☑ black box
Tester Information				
	Name of Tester:		**Date:**	
	Hardware Ver:	Robot 1.0	**Time:**	
	Setup:	A wooden turn-table should be placed on top of the cardinal direction map. This map should be aligned with a magnetic compass. There should be no metal present while the alignment is being performed. Next, the partially assembled robot should be placed on the turn-table. The MCU should be connected to a terminal to observe and record data.		

Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Write program to spool compass readings while simultaneously driving motors.	Program should be statically tested to verify accuracy. Should sample compass at a sufficient rate depending on speed.				
2	Run acceptance test	Test program should prompt user to turn the robot to an orientation and then spin the motors will then spin up and down.				
3	Plot spooled data in spreadsheet program.	Plots should be analyzed to see if compass deviated any more than 10 degrees from set point.				
	Overall test result:					

Unit testing possibilities

1. MCU (hardware)
2. LCD
3. Switches
4. Compass
5. Range finder
6. H-bridge
7. Motors
8. Chassis
9. MCU (software)

Unit Test: The Digital Compass

Module	Digital Compass – Geosensor version 2.3
Inputs	- Earth's magnetic field: An orientated field of magnetic force beginning and ending at the earth's magnetic poles. - SClk – Clock signal to clock data through the module. Maximum Frequency is 10Mhz. - SDIn – Serial data input to send data into the compass module. Date is valid on positive SClk edges.
Outputs	- SDOut – Serial data output from the compass module. Data is valid on negative clock edges.
Functionality	Senses the earth's magnetic field and determines the orientation of the compass with respect to the field. This orientation is stored in an internal register and can be retrieved through the SPI interface.
Test	Comp-UT-01

Unit test: compass (matrix)

Test Writer: Sue L. Engineer			
Test Case Name:	Compass unit test #1	**Test ID #:**	Comp-UT-01
Description:	Checks that the compass returns correct angular measurements to the MCU. Test program is in ./test/compass_unit_test_1.c	**Type:**	☐ white box ☑ black box
Tester Information			
Name of Tester:		**Date:**	
Hardware Ver:	Compass Module - Geosensor version 2.3	**Time:**	
Setup:	Compass module should be wired to the MCU through the SPI interface pins. The MCU should be connected to an RS232 terminal through its SCI interface. The terminal should be configured to run at 9600 baud. Cardinal directions map should be aligned using the magnetic compass.		

Step	Action	Expected Result	Pass	Fail	N/A	Comments
1	Compile compass.c in /test directory	IDE should generate no warnings or errors.				
2	Download	MCU should report "download successful"				
3	Execute	MCU should display compass splash screen on terminal interface.				
4	Orientate compass to 0 degrees.	Terminal interface should display 0 degrees +/- 10 degrees.				
5	Orientate compass to 30 degrees.	Terminal interface should display 30 degrees +/- 10 degrees.				
6	Orientate compass to 45 degrees.	Terminal interface should display 45 degrees +/- 10 degrees.				
…	…	…				
12	Orientate compass to 315degrees.	Terminal interface should display 315 degrees +/- 10 degrees.				
	Overall test result:					

Outline

Guidance

Reasons to develop and conduct tests

- Testing reduces the number of bugs in existing and new features.
- Tests are good documentation.
- Tests improve design.
- Tests allow you to refactor.
- Tests constrain features.
- Tests defend against other designers.
- Testing is fun.
- Testing forces you to slow down and think.
- Testing makes development faster.
- Tests reduce fear.

Summary

- Testing helps to ensure proper operation of system.
- Categories of Testing
  - Block box vs. White box (based on amount of observability)
  - Unit Testing vs. Integration Testing vs. Acceptance Testing (based on level of abstraction)
  - Matrix tests vs. Step-by-step test (based on whether the system has memory)
  - Automated tests (can apply to all test types)

Questions