DETAILED DESIGN, IMPLEMENTATIONA AND TESTING

Instructor: Dr. Hany H. Ammar

Dept. of Computer Science and Electrical Engineering, WVU

OUTLINE

- Detailed design, Implementation, and testing phase in the software development standards
- Software Testing
  - Black Box Testing
  - White Box Testing

Detailed design, Implementation, and Testing

- In the European Space Agency standard, the major activities in this phase include

  - decomposition and the specification of the low-level software modules.

    the low-level components specified at the architectural design are decomposed into software modules.

  - This is followed by software production and documentation: *implementing the modules specifications into code, integrating the software components, and the testing activities*

Detailed design, Implementation, and Testing

Decomposition of low-level components

- Using stepwise refinement, the low-level components specified at the architectural design are decomposed into software modules, and Library modules (language dependent Application Programming Interfaces APIs, e.g., Microsoft Foundation Classes, Java API packages for Graphical User Interfaces or GUIs, network programming, etc.)

Decomposition of low-level components

Example: Process Pilot Request subsystem

- Specify the detailed design of subordinate modules of Process_Pilot_Request
- Specify how buffers are structured (the data structures of the buffers used for inputs and outputs)

Detailed design, Implementation, and Testing

Software production

Software production consists of

- Implementing the modules specifications into code (structured Programming or object-oriented programming),
- Integrating the software components and the testing activities `in testing procedures of unit testing, integration testing, and system testing`

Detailed design, Implementation, and Testing

The activities required in the MIL-STD-498 standard for the Software Implementation and Unit Testing (SWIUT) are described as follows:

- Develop and document software corresponding to each software unit (CSU) of each software component (CSC) in the CSCI design
- Establish test cases (in terms of inputs, expected results, and evaluation criteria), test procedures, and test data for testing the software corresponding to each software unit

Detailed design, Implementation, and Testing

- The test cases shall cover all aspects of the unit's detailed design. The developer shall record this information in the appropriate software development files (SDFs)
- Test the software corresponding to each software unit. The testing shall be in accordance with the unit test cases and procedures

Detailed design, Implementation, and Testing

- Make all necessary revisions to the software, perform all necessary retesting, and update the software development files (SDFs) and other software products as needed, based on the results of unit testing
- Analyze the results of unit testing and record the test and analysis results in appropriate software development files (SDFs)

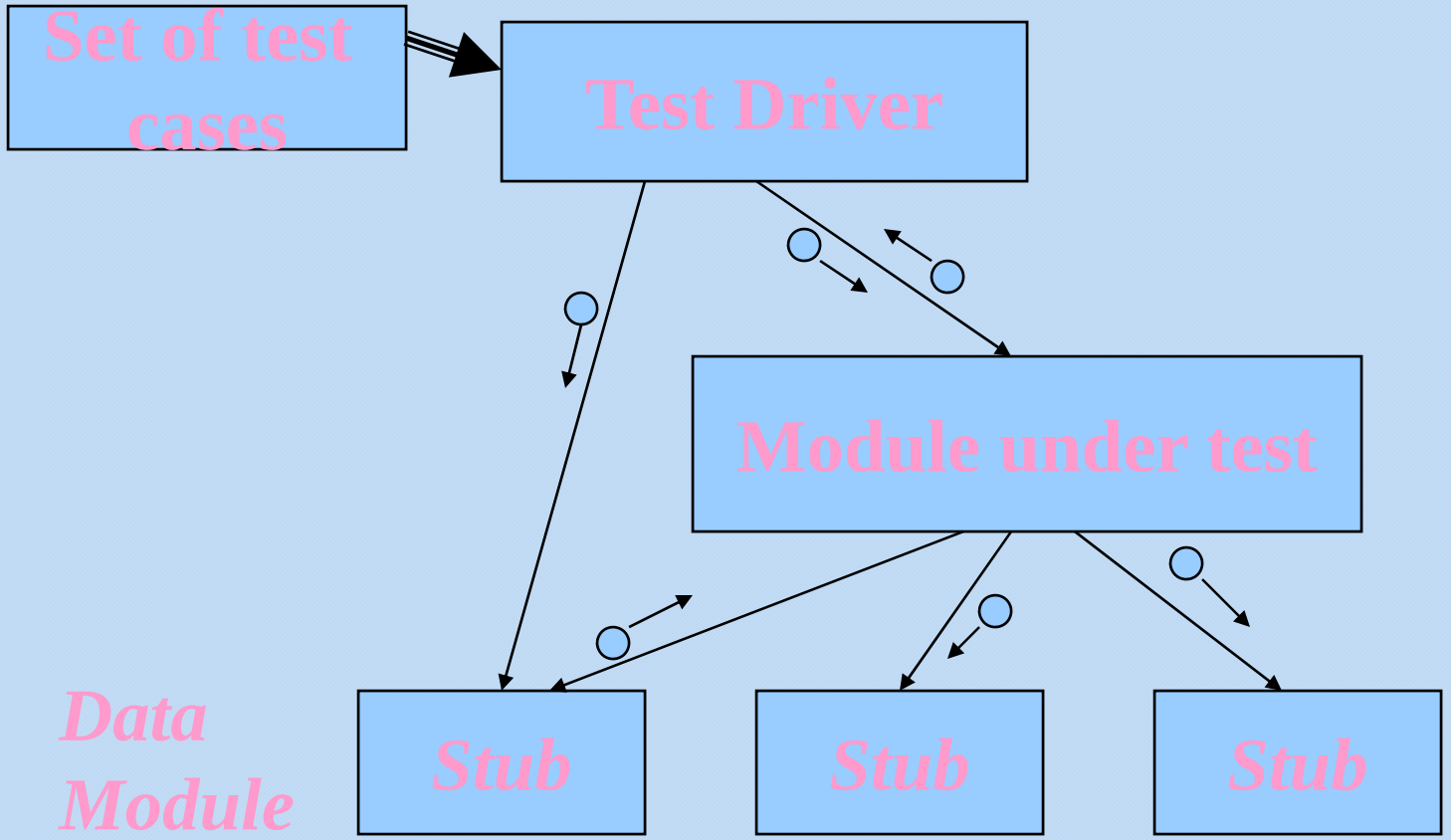
Software Testing

What is Software Testing?

- Is the process of executing a program with an established set of test cases
- Test cases are generated according to a well defined procedures or techniques
- Testing is a bottom-up process, starts with unit testing, components testing, CSCI testing

- Levels of Real Time System Testing
  - Unit Testing
  - Software Integration Testing
  - Software Validation and Verification Testing
  - Software / Hardware Integration Testin
  - System Testing

Software Testing (Unit Testing)



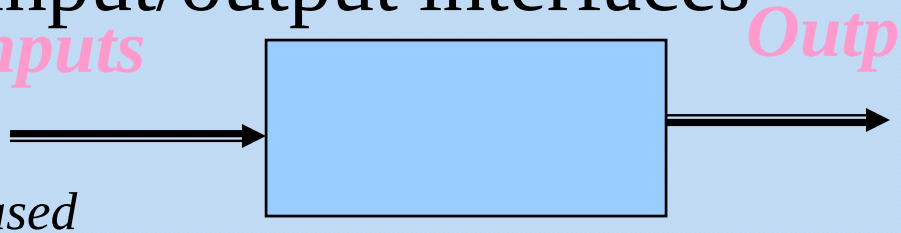
Software Testing

- The test driver is developed based on a set of test cases
- The driver invokes the module under test for each test case and establishes the test case data/control requirements
- Stubs are sub-ordinate dummy modules that represent the modules (including global data structures) invoked (or accessed) by the module under test
- They contain print and return statements

Software Testing

- Testing techniques consist of

  - Black Box testing, where we focus on testing the software functional requirements, and testing the input/output interfaces



Only inputs and outputs of functions are considered

How outputs are generated based on a set of inputs is ignored

Run a suite of test cases

-Exhaustive combination of all inputs

-Corner cases (min, max, avg)

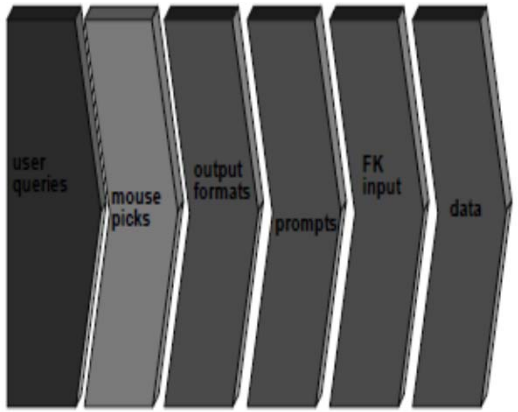
-Pathological cases (inputs likely to result in error)

Module under test Is treated as a Black Box

Software Testing

- **White Box Testing**, where we focus on developing test cases to cover the logical paths through the code (e.g., conditional statements, loops, etc.)
  - Based on developing a control flow graph of the code under test to identify the set of independent paths, and produce a set of test cases to cover these paths
  - Exercises all paths in a module Driven by logic
  - Static Example: Code Inspections, group walkthrough of software logic, inspect code line-by-line (Static Analysis tools)
  - Dynamic Example: Test all the links and buttons on a web page
  - For real-time systems dynamic testing is important to test time constraints, and reactive logical pathes

Software Testing



- Black Box Testing Techniques
  1. Equivalence Partitioning

     Partition the input space into equivalence classes and develop a test case for each class of inputs
  2. Boundary value analysis

     develop test cases at the boundaries of the possible input ranges (minimum and maximum values)
- The above techniques are important for data processing intensive applications

Black Box Testing

Sample Equivalence Classes

Valid data

- user supplied commands
- responses to system prompts
- file names
- computational data
  - physical parameters
  - bounding values
  - initiation values
- output data formatting
- responses to error messages
- graphical data (e.g., mouse picks)

Invalid data

- data outside bounds of the program
- physically impossible data
- proper value supplied in wrong place

Software Testing

- Black Box Testing Techniques (cont.)

  3. Cause-effect graphing

     - Used for control intensive applications,

     - Develops test cases to represent input events (or causes) and the corresponding actions (or effects)

- This technique is used intensively in real-time systems

Software Testing

- Cause-effect graphing consists of the following 4 steps

  1. List and label causes (input-events) and effects (output actions) for a module

  2. Draw a cause-effect graph describing the logical combinations of causes, intermediate causes and resulting effects,

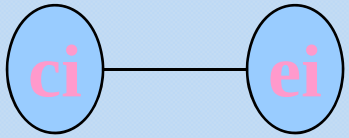
  3. Develop a decision table (causes vs effects) from the graph

  4. Convert each row into a test case, or a set of rows into a testing scenario

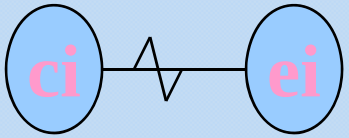
Software Testing

Symbols used in the cause-effect graphs

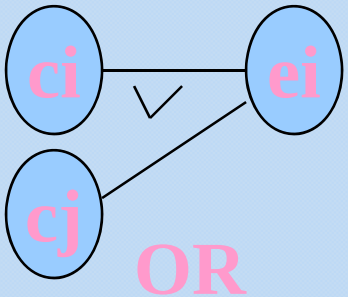
ci = ith cause, ei = ith effect



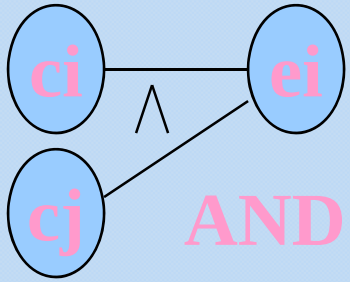
Ci causes ei



Ci does Not cause ei



OR



AND

Software Testing

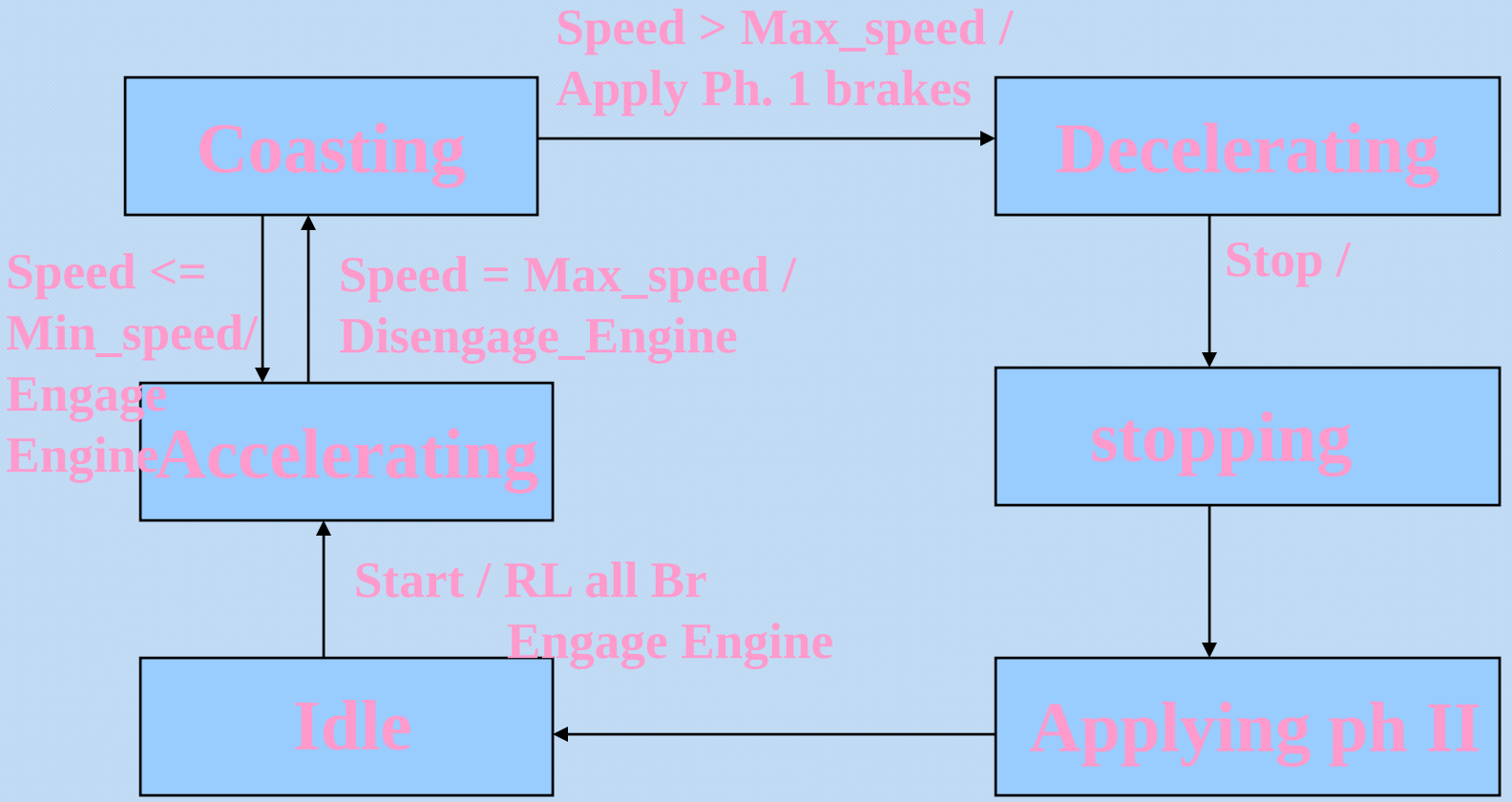
- Example: Control motion module in our project (assume the module inputs are a state variable and a control input)

1. List and label all Causes and all effects

Causes	Effects
C1: Start_train	e100: Release all Brakes
C2: Idle	e200: Engage_Engine
C3: speed = Max_speed	e300: Disengage_Engine
C4:Accelerating	e400: Apply Ph. 1 Brakes
C5: Speed > Max_speed	e500: Release Ph. 1 brakes
C6: Coasting	e600: Apply Ph. 2 brakes
C7: Speed <= Min_speed	e700: Apply Safety brakes

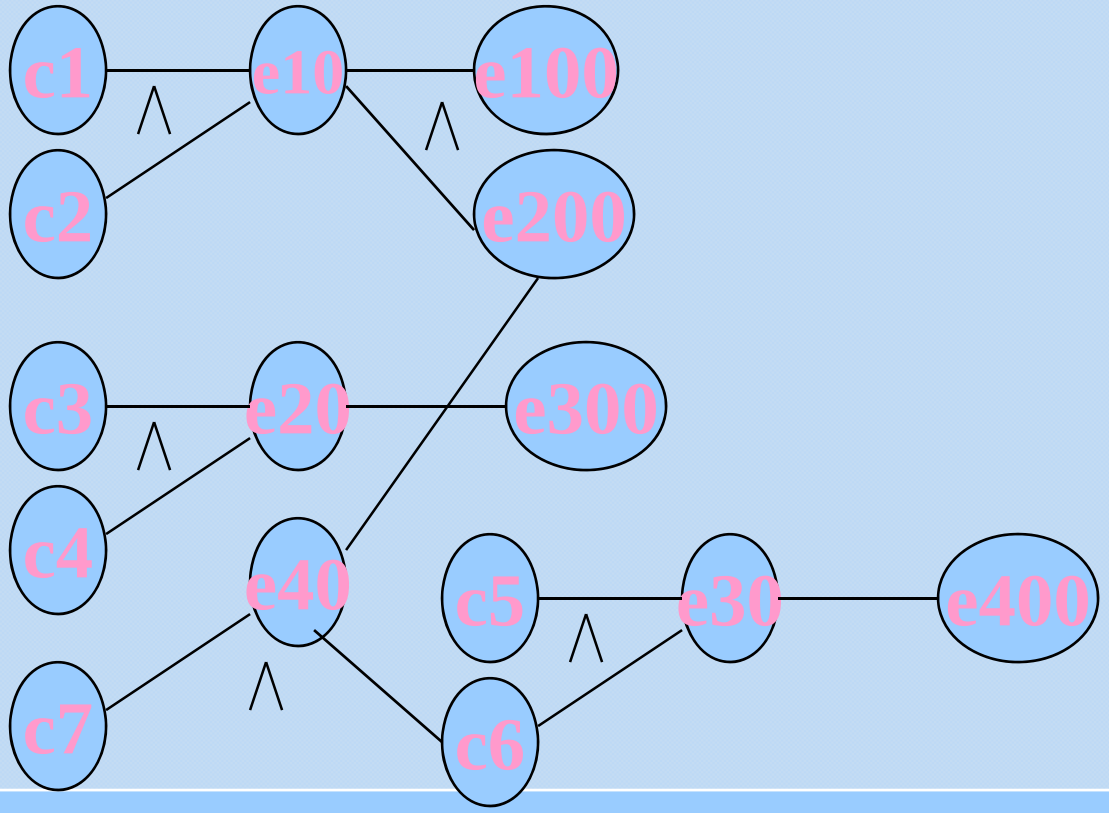
Software Testing

This List is obtained form the following C-spec STD



Software Testing

Develop a Cause effect graph



Software Testing

- Develop a Decision Table

C1	C2	C3	C4	C5	C6	C7	e100	e200	e300	e400
1	1	0	0	0	0	0	1	1	0	0
0	0	1	1	0	0	0	0	0	1	0
0	0	0	0	1	1	0	0	0	0	1
0	0	0	0	0	1	1	0	1	0	0

- Convert each row to a test case

Software Testing

- Scenario testing: A testing scenario combines a set of test cases for testing the system behavior over a period of time
- For example the following sample run can be used for scenario testing

Start,idle ; speed = max_speed, Accelerating;

Speed < = Min_speed, Coasting; Stop, Accelerating; speed = stop speed; start, Idle; speed = max_speed, Accelerating; speed > Max_speed, Coasting;