Informatics 43

LECTURE 8-1

EMILY NAVARRO

Last Time

- Many failures are caused by a lack of good quality assurance (QA) practices
- QA = All activities designed to measure and improve quality in a product
  - Validation & verification
- Testing is the most common QA activity
  - Different levels: unit/functional/system
  - Goal: find and fix failures/faults/errors
  - Can never be exhaustive
  - Can never prove a system's correctness
- All software has bugs!

Today's Lecture

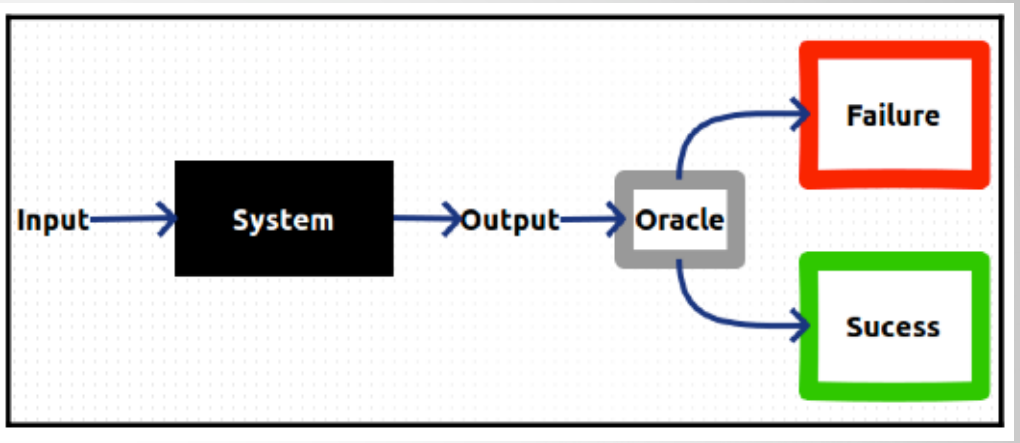
- Black-box (Specification-based) Testing
- Homework 2

Today's Lecture

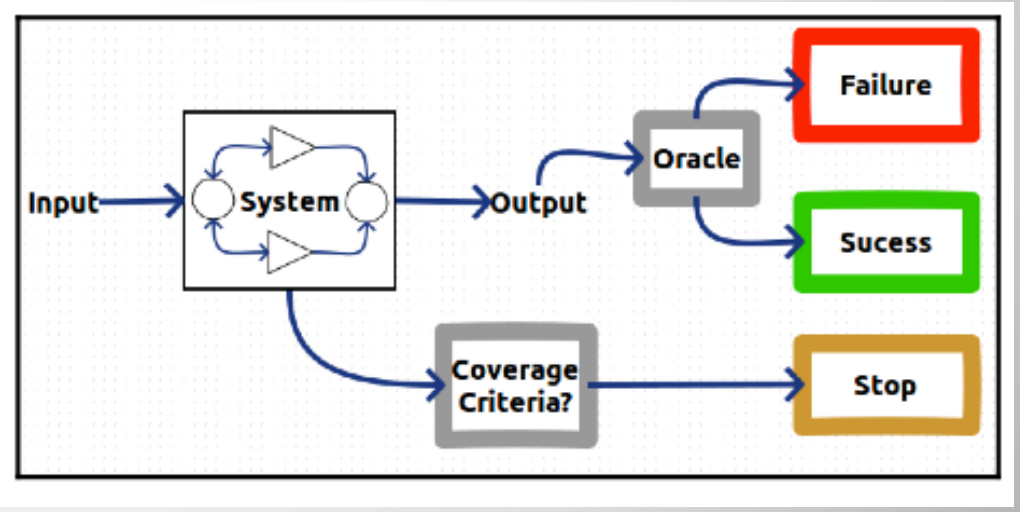
- Black-box (Specification-based) Testing
- Homework 2

Two Overall Testing Approaches

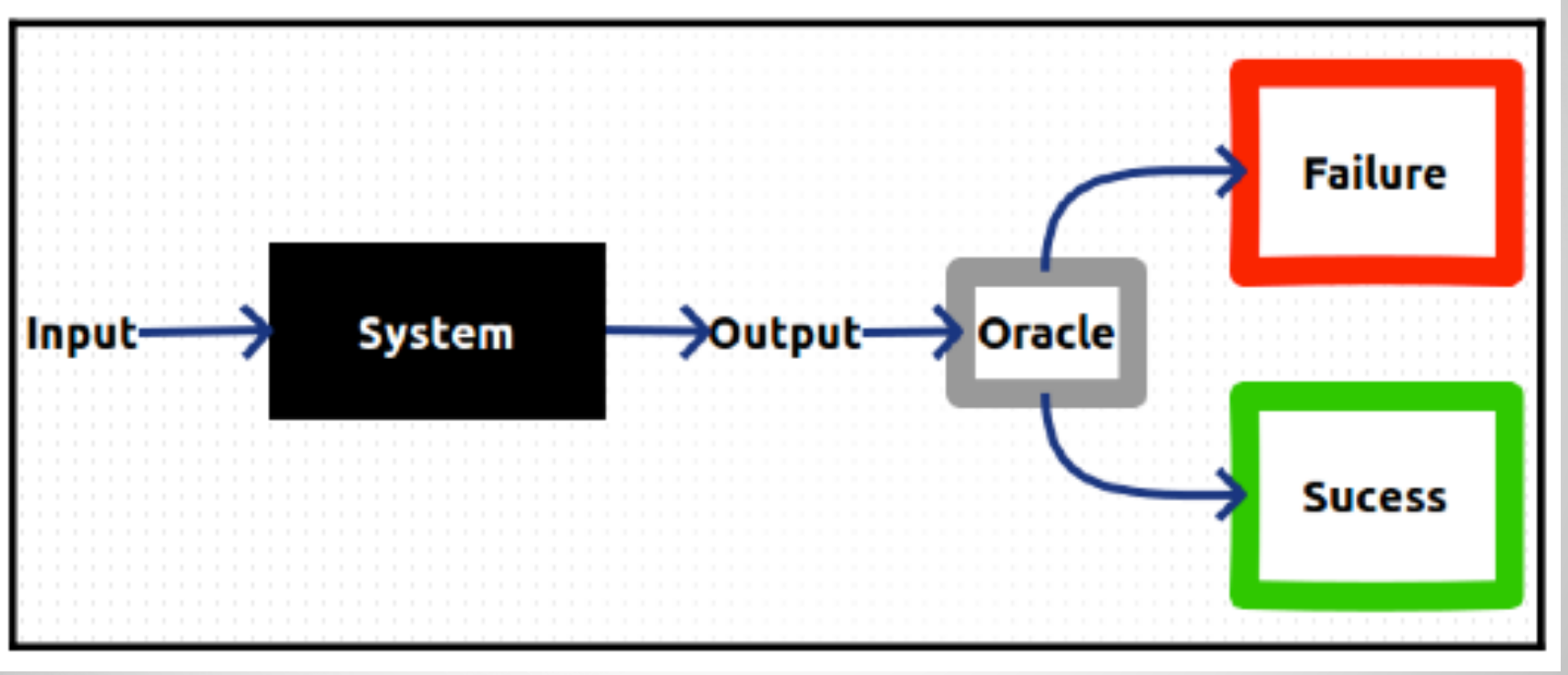
- Black box testing
  - Specification-based testing



- White box testing
  - Structural testing



Black-box/Specification-Based Testing



Black-box/Specification-Based Testing

- Use specifications to derive test cases
- Choose test cases that guarantee a wide range of coverage
  - Typical values
  - Boundary values
  - Special cases
  - Invalid input values

Equivalence Class Partitioning

- Divide the input into several *classes* that are considered "equivalent" for the purposes of finding errors
  - If it fails/passes for one member of the class, it is likely to fail/pass for all members
- Classes are determined by
  - Looking at the requirements specification
  - Tester's intuition
- Classes
  - **should cover the complete input domain**
  - **should never overlap**

Boundary Value Analysis

- Experience has shown that many errors are made at the "boundaries" rather than under normal conditions
  - E.g., confusion between < and <=
- Boundary value analysis uses the same classes as equivalence partitioning, but tests at the boundaries of the classes, rather than just any element from the class

Equivalence Class Partitioning/Boundary Value Analysis: A Systematic Approach

1. Identify the set of all possible inputs (to what is being tested), aka "domain"
2. Identify a basis for subdividing the set of inputs
   - Possible bases
     - Size/magnitude
     - Structure
     - Correctness
     - Your creative thinking

Equivalence Class Partitioning/Boundary Value Analysis: A Systematic Approach (cont.)

3. Use this basis to divide the set of all possible inputs into classes/subdomains
4. From each subdomain, select [a] representative(s) to be [a] test case input(s)
   - One test case may suffice
5. Test for each subdomain
   - "Normal" values
   - Boundary or edge input values (Boundary Value Analysis)

Example: quizAverage

- Input: A list of numbers (integers)
  - Scores must be between 0 and 100 (inclusive)
- Output: a single number which is the average of the numbers on the input list, not counting the lowest number on the list.

Example: quizAverage

1. Identify the set of all possible inputs (to what is being tested)
2. Identify a basis for subdividing the set of inputs
3. Use this basis to divide the set of all possible inputs into subdomains
4. From each subdomain, select [a] representative(s) to be [a] test case input(s)

Example: quizAverage

1. Identify the set of all possible inputs (to what is being tested)

   The set of all lists of integers

2. Identify a basis for subdividing the set of inputs
3. Use this basis to divide the set of all possible inputs into subdomains
4. From each subdomain, select [a] representative(s) to be [a] test case input(s)

Example: quizAverage

1. Identify the set of all possible inputs (to what is being tested)

   The set of all lists of integers

2. Identify a basis for subdividing the set of inputs

   length of the list, position of minimum score, number of minima, magnitude of numbers

3. Use this basis to divide the set of all possible inputs into subdomains

4. From each subdomain, select [a] representative(s) to be [a] test case input(s)

Example: quizAverage

1. Identify the set of all possible inputs (to what is being tested)

   The set of all lists of integers

2. Identify a basis for subdividing the set of inputs

   length of the list, position of minimum score, number of minima, magnitude of numbers

3. Use this basis to divide the set of all possible inputs into subdomains

   0 elements, 1, 2-10, 11+, first, middle, last, 1, a few, all, 0-33, 34-66, 67-100, mixed

4. From each subdomain, select [a] representative(s) to be [a] test case input(s)

Example: quizAverage

1. Identify the set of all possible inputs (to what is being tested)

   The set of all lists of integers

2. Identify a basis for subdividing the set of inputs

   length of the list, position of minimum score, number of minima, magnitude of numbers

3. Use this basis to divide the set of all possible inputs into subdomains

   0 elements, 1, 2-10, 11+, first, middle, last, 1, a few, all, 0-33, 34-6, 67-100, mixed

4. From each subdomain, select [a] representative(s) to be [a] test case input(s)

   [], [87.3], [90,95,85], [80,81,82,83,84,85,86,87,88,89,90,91],

   [80,87,88,89], [87,88,80,89], [87,88,89,80],

   [80,87,88,89], [87,86,86,88], [88,88,88,88],

   [0,4,15,5,33], [47, 43, 58, 60, 34], [100, 67], [0, 35, 99]

Possible Basis: List Length

- Input domain: all possible lists of integer numbers
- Subdomains
  - Empty list []
  - One element [87]
  - Small (two to ten elements) [90,95,85]
  - Large (eleven+ elements) [80,81,82,83,84,85,86,87,88,89,90,91]

Possible Basis: Position of Minimum Score

- Input domain: all possible lists of integer numbers
- Subdomains
  - Smallest element first [80,87,88,89]
  - Smallest element in middle [87,88,80,89]
  - Smallest element last [87,88,89,80]

Possible Basis: Number of Minima

- Input domain: all possible lists of integer numbers
- Subdomains
  - One minimum [80,87,88,89]
  - A few minima [87,86,86,88]
  - All minima [88,88,88,88]

Possible Basis: Magnitude of Numbers

- Input domain: all possible lists of integer numbers
- Subdomains
  - Small (0-33) [0,4,15,5,33]
  - Medium (34-66) [47, 43, 58, 60, 34]
  - Large (67-100) [100, 67]
  - Mixed [0, 35, 99]

Testing Matrix

Basis

Test case (input)	Subdomain	Expected output	Actual output

quizAverage 1

Basis: List length

Test case (input)	Subdomains				Expected output	Actual output
	Empty	One	Small	Large		
()	x				0.0	99999!
(87)		x			87.0	crashes!
(90,95,85)			x		92.5	92.5
(80,81,82,83, 84,85,86,87, 88,89,90,91)				x	86.0	86.0

quizAverage 2

Basis: Position of minimum

Test case (input)	Subdomains			Expected output	Actual output
	First	Middle	Last		
(80,87,88,89)	x			88.0	88.0
(87,88,80,89)		x		88.0	88.0
(99,98,0,97,96)		x		97.5	97.5
(87,88,89,80)			x	88.0	88.0

quizAverage 3

Basis: Number of minima

Test case (input)	Subdomains			Expected output	Actual output
	One	Several	All		
(80,87,88,89)	x			88.0	88.0
(87,86,86,88)		x		87.0	87.0
(99,98,0,97,0)		x		73.5	98.0
(88,88,88,88)			x	88.0	88.0

quizAverage 4

Basis: Magnitude of numbers

Test case (input)	Subdomains				Expected output	Actual output
	0-33	34-66	67-100	Mixed		
(0,4,15,5,33)	x				14.25	14.25
(47,43,58,60,34)		x			52	52
(100,67)			x		100	100
(0, 35, 99)				x	67	67

Example: Hotel Management System

- Consider a hotel management system that takes **phone numbers** as input while gathering data about the guest
- Imagine we want to test the "input phone number" function of the system
- Specification: Should give a descriptive error message if
  - input is less than 10 digits
  - input is more than 20 digits
  - input contains non-numeric characters
- What are the properties about phone numbers that we can exploit to create "valuable" partitions?

Input Phone Number 1

Basis: length of input

Subdomains	Test Case Input Data	Expected Output
Too small (<10 digits)	3334444 333-4447	error error
Too large (>20 digits)	283948582930900445554	error
Valid small (10-15 digits long)	8849394858	ok
Valid large (16-20 digits long)	8283333849573849	ok

Input Phone Number 2

Basis: content of input

Subdomains	Test Case Input Data	Expected Output
Contains only numeric characters	2938485938 283948574859384739	Ok ok
Contains non-numeric characters (but not (,),-,#,*,+,:	1234b8983 Gieoi!^%@	Error error
Contains (,),-,#,*,+,:	(949)824-6300	error

Input Phone Number 3

Basis: position of invalid characters

Subdomains	Test Case Input Data	Expected Output
beginning	(9498246300	error
middle	949824-6300	error
end	9498246300-	error
No invalid characters	9498246300	ok
Multiple places	(949)824-6300-	error

Example: Email

- Imagine we are testing the login functionality of an email program
  - Input: username, password
  - Output: login successful or error message
- Two users:
  - Mary; maryspassword
  - Joe; joespassword
- What possible bases can we use to divide our testing into partitions?

Login

Basis: whether or not the password matches

Subdomains	Test Case Input Data	Expected Output
Password matches user	Mary, maryspassword	ok
Password matches another user	Mary, joespassword	error
Password matches no user	Mary, nopassword	error

Example: Room Scheduler System

- Imagine we are testing a classroom scheduler program that handles M-F scheduling for five classrooms
- Room capacities
  - Room A: 500
  - Room B: 300
  - Room C: 100
  - Room D: 50
  - Room E: 20
- All classes are 1-hour long, once per week, and can be scheduled between 8am-10pm

Example: Room Scheduler System

- Input
  - The current schedule
  - The number of students in the class to be scheduled
  - The desired time of the class to be scheduled
- Output
  - A list of available rooms that can hold the number of students, ordered from most appropriate (number of students is as close as possible to room capacity without going over) to least appropriate

Example: Room Scheduler System

- Example
  - Input:
    - {Current schedule:
    - Room A: M-F: 8-11am, 2-4pm
    - Room B: T-F: 9-10am, 5-8pm
    - Room C: F: 10am-3pm
    - Room D: M-F: 8am-10pm
    - Room E: M-F: 10am-5pm, 8pm-10pm;
    - Num students: 73
    - Desired time: W 5-6pm}
  - Expected output: {Room C, Room A}

Room capacities
Room A: 500
Room B: 300
Room C: 100
Room D: 50
Room E: 20

Example: Room Scheduler System

- What possible bases can we use to divide our testing into partitions?
  - Fullness of schedule
  - Number of students in the class to be scheduled
  - Desired time

Schedule Room 1

Basis: Fullness of schedule

Subdomains	Test Case Input Data	Expected Output
Empty schedule	EMPTY_SCHEDULE, 150, T 5-6pm	Room B, Room A
Partially full schedule	SCHEDULE_A, 73, W 5-6pm	Room C, Room A
Full schedule	FULL_SCHEDULE, 425, F 9-10am	No rooms available

Schedule Room 2

Basis: Class size

Subdomains	Test Case Input Data	Expected Output
Invalid input	SCHEDULE_A, -5, T 5-6pm	Error
Small (0-20)	SCHEDULE_A, 3, M 2-3pm	Room C, Room B
Medium (21-199)	SCHEDULE_A, 25, M 2-3pm	
Large (200-500)	SCHEDULE_A, 250, T 1-2pm SCHEDULE_A, 500, M 8-9pm	Room B, Room A Room A
Over capacity (500+)	SCHEDULE_A, 50000, F 3-4pm	Error

Schedule Room 3

Basis: Desired time

Subdomains	Test Case Input Data	Expected Output
Morning	SCHEDULE_A, 55, W 8-9am	
Afternoon	SCHEDULE_A, 400, T 1-2PM	
Evening	SCHEDULE_A, 250, M 6-7PM	

Summary

- Black box testing uses specifications to derive test cases
- We use equivalence class partitioning and boundary value analysis to choose test cases that guarantee a wide range of coverage
  - Typical values, boundary values, special cases, invalid input

Today's Lecture

- Black-box (Specification-based) Testing
- Homework 2

Homework 2

- You will be designing test cases for HMM using a black-box/specification-based approach
  - You will be provided with a specification upon which to base your testing
  - Your document will mainly consist of testing matrices
- Homework 2 is posted
- Deadline is Tuesday, May 24, 11:55pm to EEE

What are some possible bases for HW2?

- UC1
  - Length of description
  - Zip code length
  - Zip code value
  - Platform
- UC2
  - Size of TA list / number of available Tas
  - Number of TAs equidistant from customer
  - Distance between customer and closest TA
  - Number of Tas whose expertise match the device/platform of the ticket

Next Time

- White-box (Structural) Testing