# Monday, March 2



# Overview

- Last Class
  - Testing Review
- This Class
  - Testing
  - JUnit
- Next Class
  - Testing

# Testing Objectives

- Goal of testing is to make the software misbehave
  - Failures tell you a lot more than successes
- Your reward is finding a bug, even if it's your own code
  - No prizes for test cases that pass
- Testing can only tell you about the presence of defects
  - Need to use proofs and other checks to show correctness



# The Tester's Role on Agile Projects

| Testers in their traditional role | Tester role in an agile project |
|---|---|
| ■ A separate QA group<br>■ Tests are derived from detailed requirements and specifications<br>■ QA may or may not participate in planning sessions, but is not usually informed about design considerations until after they have been finalized | ■ Is part of the team and attends all team sessions<br>■ Is an integral part of the planning game<br>■ Practices pair testing, i.e. collaborates with the developers to get good tests |

http://www.ucalgary.ca/~ageras/wshop/abstracts/2003/role-agile-tester.htm

# Test Planning

- Quality Criteria
  - What are you testing for?
- Techniques
  - How are you going to test?
- Sufficiency Criteria
  - How many test cases are enough?





# Quality Factors

| | |
|---|---|
| Product Operation | |
| •Correctness | Does it do what I want? |
| •Reliability | Does it do it accurately all of the time? |
| •Efficiency | Will it run my hardware as well as it can? |
| •Integrity | Is it secure? |
| •Usability | Can I run it? |
| Product Revision | |
| •Maintainability | Can I fix it? |
| •Testability | Can I test it? |
| •Flexibility | Can I change it? |
| Product Transition | Will I be able to… |
| •Portability | …use it on another machine? |
| •Reusability | …reuse some of the software? |
| •Interoperability | …interface it with another system? |

## Quality Assurance Activities

- Verification
  - Check product against specification
  - Building the system right
- Validation
  - Check product against world (stakeholder expectations)
  - Building the right system
- van Vliet considers all quality assurance activities as testing





## What are good tests?

- Depends on the quality factor
  - Some quality factors are hard to test for and need to be built in
- Most basic are correctness and reliability
  - Others are more specialized
- Need some for success scenarios
  - Singly and in combination
- Need many for extensions, alternatives, and exceptions
  - Majority of the work (thinking and typing)

# Automated Testing

- Idea: Testing is repetitive. Get a computer to do the work for you.
  - Computers are good at repetitive sequences and don't get bored.
  - More reliable and robust than testing by hand.
- Benefits
  - Can test frequently at little additional cost
  - Greater confidence in the code
- Costs
  - Tests need to maintained along with code
    - e.g. refactoring



# Testing Techniques

- Different tactics for revealing bugs in the code

# Testing Techniques

- Manual Test Techniques
  - Reading
  - Walkthroughs and Inspections
  - Stepwise Abstraction
- Scenario-Based Evaluation
- Correctness Proofs
- Coverage-Based Techniques
- Fault-Based Techniques
- Error-Based Techniques





# Wednesday, March 3

# Overview

- Last Class
  - Testing
- This Class
  - Testing
- Next Class
  - Testing





# Testing Techniques

- Manual Test Techniques
  - Reading
  - Walkthroughs and Inspections
  - Stepwise Abstraction
- Scenario-Based Evaluation
- Correctness Proofs
- Coverage-Based Techniques
- Fault-Based Techniques
- Error-Based Techniques

# Definitions

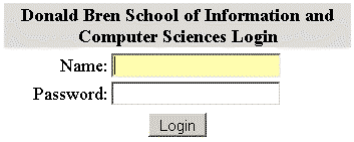
- Error
  - A human action that produces an incorrect result
- Fault
  - Manifestation of an error
  - The result of an error is a fault in the code.
- Failure
  - Observable consequences of a fault or faults
- A failure may be caused by more than one fault and a fault may cause different failures.



# Error-Based Techniques

- Certain kinds of problems are known to be difficult
  - Lead to common errors
  - Go after these errors

# Security Bugs



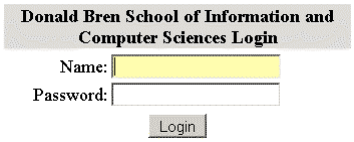

```
SELECT * FROM users
WHERE username = 'NAME'
and password = 'PASSWORD'
```

**Name:** a
**Password:** ' OR 't'='t

```
SELECT * FROM users
WHERE
username = 'a' and password = ''
OR 't'='t'
```



# Security Bugs




```
SELECT * FROM users
WHERE username = 'NAME'
and password = 'PASSWORD'
```

**Name:** a
**Password:** ';DROP TABLE users;--

```
SELECT * FROM users
WHERE username = 'a' and password = '';
DROP TABLE users;
--'
```

# Where do Bugs Hide?

- Escaping characters when converting between encoding schemes
  - HTML <-> Java <-> Database

Java - HTML

```
// HTML Special Chars
  if (c == '"')
      sb.append("&quot;");
  else if (c == '&')
      sb.append("&amp;");
  else if (c == '<')
      sb.append("&lt;");
  else if (c == '>')
      sb.append("&gt;");
  else if (c == '\n')
      // Handle Newline
      sb.append("&lt;br/&gt;");
```

Java - SQL

```
SELECT * FROM Books WHERE title = 'INPUT'

INPUT: Yasser's Book


SELECT * FROM Books WHERE title = 'Yasser's
    Book'
```



# Where do Bugs Hide?

- Threads / Concurrency
  - When you get different results on the same input
  - Testing and debugging multithreaded programs is difficult because concurrency hazards do not manifest themselves uniformly or reliably.

**3:47.01** User 1 initiates tx to pay electricity online

- Select in DB
- Account number: 123
- OldBalance: $30

**3:47.03** User 2 deposits $5,000

- Update in DB
- Account number: 123
- OldBalance: $30
- Balance: $5,030

**3:47.05** User 1 finishes tx to pay electricity online $10

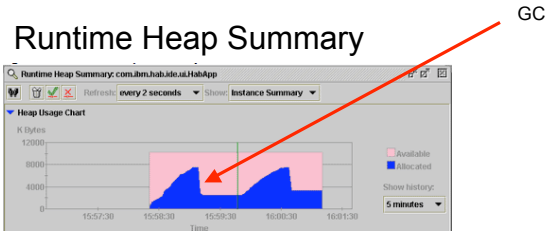
- Update in DB
- Account number: 123
- OldBalance: $30
- Balance: $20

- Recommendation: Verify that the information did not change in DB before updating it

# Where do Bugs Hide?

- Memory
  - Be careful about memory leaks in Java.
  - The job of the garbage collector is to find objects that are no longer needed by an application and to remove them when they can no longer be accessed or referenced.
  - The key point to remember is that an object is only counted as being unused when it is no longer referenced.
  - If your program is getting a java.lang.OutOfMemoryError after executing for a while, a memory leak is highly likely.

Runtime Heap Summary






# Where do Bugs Hide?

- Preventing memory leaks
  - Collection classes, such as **hashtables and vectors**, are common places to find the cause of a memory leak. Specially if the class has been declared **static** and exists for the life of the application.
  - Another common problem occurs when you register a class as an **event listener** without bothering to unregister when the class is no longer needed.
  - Member variables of a class that point to other classes simply **need to be set to null at the appropriate time**.

# Common Time and Date Errors

- Y2K
- mm/dd/yyyy (the US) vs dd/mm/yyyy (rest of the world
- 24 hour vs. 12 hour clock
- Formatting
  - Day 36 of a month?
  - Hour 27 of a day?
- Leap Years



# Friday, March 6

# Overview

- Last Class
  - Testing
- This Class
  - Testing
- Next Class
  - Testing



# Common Time and Date Errors

- Time Zones
- Daylight Saving Time vs. Standard Time
  - Some locations change some don't
  - Exact time and date of change
  - Missing and extra hours
  - Northern hemisphere vs. Southern hemisphere

# Announcements

- No lab next week
- No discussion next week

- Course evaluations
  - If we have 75% participation by Friday, I will do review
  - Otherwise, we will do course evaluations in class
    - And possibly cover new material





# Quality Assurance Activities

- Verification
  - Check product against specification
  - Building the system right
- Validation
  - Check product against world (stakeholder expectations)
  - Building the right system

- van Vliet considers all quality assurance activities as testing

# Coverage-Based Techniques

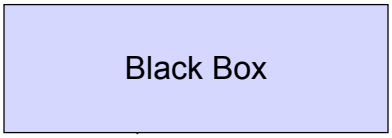
- Coverage is expressed in terms of how much of the software work product has been covered by testing
  - Percentage of statements, paths, branches, etc.
  - Percentage of requirements
- Control-Flow Coverage
- Data-Flow Coverage
- Both are based on turning the work product into a graph





# How Do We Choose Test Cases?

```
public boolean isValidMonth(int num) {
```




```
}
```

- Can we test this function for $-2^{31}$ to $2^{31}-1$?
- Is there any difference between *isValidMonth(13)*, *isValidMonth(100)*, *isValidMonth(1000)*?

# Equivalence partitioning

- Input data and output results often fall into different classes where all members of a class are related e.g. positive numbers, negative numbers, strings without blanks, etc.
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition- input and outputs lie within partitions

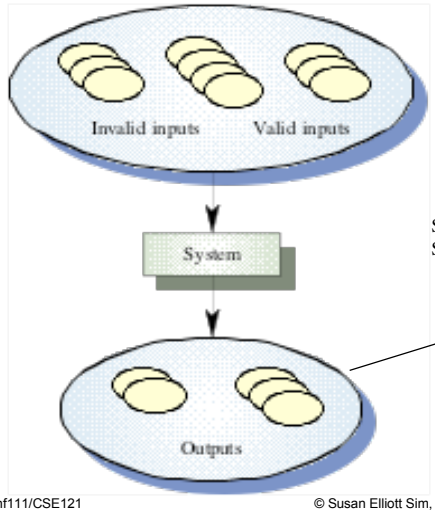


# Example: Age of Customers

| Class | Representative |
|---|---|
| Low | -5 |
| 0-12 | 6 |
| 13-19 | 15 |
| 20-35 | 30 |
| 36-120 | 60 |
| High | 160 |

# Equivalence partitioning

Each equivalence partition is shown as an ellipse.




Input equivalence partitions

Sets of data where all the set members
Should be processed in equivalent way

Output equivalence partitions

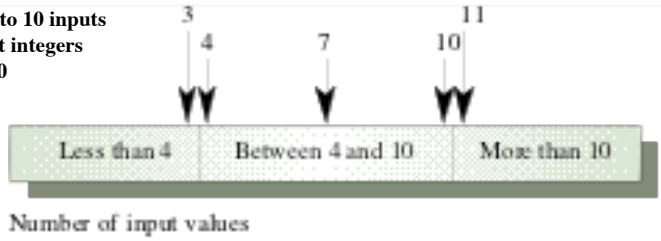
Program outputs that have
common characteristics



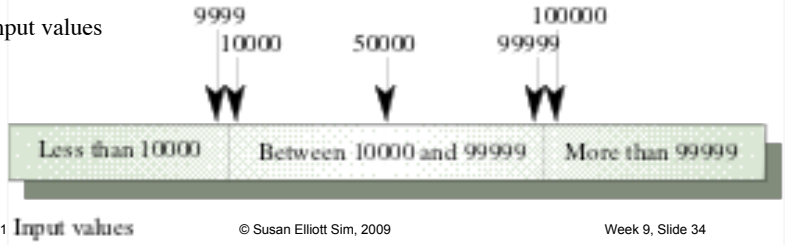


# Equivalence partitions

**Program accepts 4 to 10 inputs which are five- digit integers greater than 10,000**




Possible test input values



# Search routine specification

**procedure** Search (Key : ELEM ; T: ELEM_ARRAY;
Found : **out** BOOLEAN; L: **out** ELEM_INDEX) ;

**Pre-condition**
The array has at least one element
T'FIRST <= T'LAST
**Post-condition**
The element is found and is referenced by L
( Found and T (L) = Key)
**or**
The element is not in the array
( **not** Found **and**
**not** (**exists** i, T'FIRST >= i <= T'LAST, T (i) = Key ))



# Search routine - input partitions

- Inputs where the key element is a member of the array
- Inputs where the key element is not a member of the array
- Inputs where a pre-condition does not hold

# Search routine - input partitions

| **Array** | **Element** |
|---|---|
| Single value | In sequence |
| Single value | Not in sequence |
| More than 1 value | First element in sequence |
| More than 1 value | Last element in sequence |
| More than 1 value | Middle element in sequence |
| More than 1 value | Not in sequence |

| **Input sequence** (T) | **Key** (Key) | **Output** (Found, L) |
|---|---|---|
| 17 | 17 | true, 1 |
| 17 | 0 | false, ?? |
| 17, 29, 21, 23 | 17 | true, 1 |
| 41, 18, 9, 31, 30, 16, 45 | 45 | true, 7 |
| 17, 18, 21, 23, 29, 41, 38 | 23 | true, 4 |
| 21, 23, 29, 33, 38 | 25 | false, ?? |