Announcements

- Second assignment out later today
  - Topic: Code Inspection and Testing
    - Find defects in Hnefetafl rules written by your classmates
    - Compare inspection, coverage testing, random testing, and black-box testing
  - Group assignment
  - Due Wednesday, January 28, at 10:30am
- TA Introductions
  - Taekgoo Kim
  - Darpan Saini

Testing

Primary source: Kaner, Falk, Nguyen.
Testing Computer Software (2nd Edition).

Jonathan Aldrich
Assistant Professor
Institute for Software Research

School of Computer Science
Carnegie Mellon University

jonathan.aldrich@cs.cmu.edu
+1 412 268 7278

Testing – The Big Questions

1. **What is testing?**
   - And why do we test?
2. **To what standard do we test?**
   - Specification of behavior and quality attributes
3. **How do we select a set of good tests?**
   - Functional (black-box) testing
   - Structural (white-box) testing
4. **How do we assess our test suites?**
   - Coverage, Mutation, Capture/Recapture…
5. **What are effective testing practices?**
   - Levels of structure: unit, integration, system…
   - Design for testing
   - How does testing integrate into lifecycle and metrics?
6. **What are the limits of testing?**
   - What are complementary approaches?
     - *Inspections*
     - *Static and dynamic analysis*

1. Testing: What and Why

- What is testing?
  - Direct execution of code on test data in a controlled environment
- Discussion: Goals of testing

1. Testing: What and Why

- What is testing?
  - Direct execution of code on test data in a controlled environment
- Discussion: Goals of testing
  - To reveal failures
    - Most important goal of testing
  - To assess quality
    - Difficult to quantify, but still important
  - To clarify the specification
    - Always test with respect to a spec
    - Testing shows inconsistency
      - Either spec or program could be wrong
  - To learn about program
    - How does it behave under various conditions?
    - Feedback to rest of team goes beyond bugs
  - To verify contract
    - Includes customer, legal, standards

Testing is NOT to show correctness

- Theory: "Complete testing" is impossible
  - For realistic programs there is always untested input
  - The program may fail on this input
- Psychology: Test to find bugs, not to show correctness
  - Showing correctness: you fail when program does
  - Psychology experiment
    - People look for blips on screen
    - They notice more if rewarded for finding blips than if penalized for giving false alarms
  - Testing for bugs is more successful than testing for correctness
    - [Teasley, Leventhal, Mynatt & Rohlman]

Testing – The Big Questions

1. **What is testing?**
   - And why do we test?
2. **To what standard do we test?**
   - Specification of behavior and quality attributes
3. **How do we select a set of good tests?**
   - Functional (black-box) testing
   - Structural (white-box) testing
4. **How do we assess our test suites?**
   - Coverage, Mutation, Capture/Recapture…
5. **What are effective testing practices?**
   - Levels of structure: unit, integration, system…
   - Design for testing
   - How does testing integrate into lifecycle and metrics?
6. **What are the limits of testing?**
   - What are complementary approaches?
     - *Inspections*
     - *Static and dynamic analysis*

Specifications

- Contains
  - Functional behavior
  - Erroneous behavior
  - Quality attributes
- Desirable attributes
  - Complete
    - Does not leave out any desired behavior
  - Minimal
    - Does not require anything that the user does not care about
  - Unambiguous
    - Fully specifies what the system should do in every case the user cares about
  - Consistent
    - Does not have internal contradictions
  - Testable
    - Feasible to objectively evaluate
  - Correct
    - Represents what the end-user(s) need

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me exactly $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility

- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running

- Example:

```
/*@ requires array != null && len >= 0 && array.length == len
  @
  @ ensures \result == (\sum int j; 0<=j && j<array.length;  array[j])
  @*/
public float sum(int array[], int len) {… }
```

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me exactly $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility

- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running

- Example:

```
/** Applies a move to a board. This assumes that the move is one that
   was returned by getAllMoves. Upon applying the move, it will also
   update the value of the board and switch the board's turn.  */
public void applyMove(Move mv) { … }
```

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me exactly $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility
- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running
- (Functional) correctness with respect to the specification
  - If the client of a function fulfills the function's precondition, the function will execute to completion and when it terminates, the postcondition will be fulfilled
- What does the implementation have to fulfill if the client violates the precondition?

Function Specifications

- A function's contract is a statement of the responsibilities of that function, and the responsibilities of the code that calls it.
  - Analogy: legal contracts
    - If you pay me exactly $30,000
    - I will build a new room on your house
  - Helps to pinpoint responsibility
- Contract structure
  - Precondition: the condition the function relies on for correct operation
  - Postcondition: the condition the function establishes after correctly running
- (Functional) correctness with respect to the specification
  - If the client of a function fulfills the function's precondition, the function will execute to completion and when it terminates, the postcondition will be fulfilled
- What does the implementation have to fulfill if the client violates the precondition?
  - A: Nothing. It can do anything at all.

Quick Quiz

Assume the specification for sum given in the lecture slides:

requires array != null && len >= 0 && array.length == len
ensures \result == (\sum int j; 0 <= j && j < len; array[j])

Assume the following input and outputs for sum, where a 3 element array is written as [1, 2, 3]. For which of the inputs and outputs is the implementation of sum correct according to the specification given?

- Input: array = [1, 2, 3, 4], len = 4
  Output: 10
- Input: array = [0, 0, 3, -7], len = 4
  Output: *none (the program does not terminate)*
- Input: array = [1, 2, 3, 4], len = 3
  Output: 7
- Input: array = [1, 2, -3, 4], len = 4
  Output: 7

Erroneous Behavior Specifications

- A function can do anything at all if precondition is violated, BUT…
  - we may want the system to function even if one part fails
  - we may want to easily identify our mistakes
- Exceptional case specifications
  - Precondition: condition describing the input that leads to an error
  - Postcondition: condition established by the function under that erroneous input
- Example (BitSet.toArray() in JML)

```
/*@ public normal_behavior
    @   requires a!= null;
    @   requires (\forall Object o; containsObject(o);
    @                               \typeof(o) <: \elemtype(\typeof(a)));
    @ also
    @ public exceptional_behavior
    @   requires a == null;
    @   signals_only NullPointerException ;
    @ also
    @ public exceptional_behavior
    @   requires a != null;
    @   requires !(\forall Object o; containsObject(o);
    @                               \typeof(o) <: \elemtype(\typeof(a)));
    @   signals_only ArrayStoreException ;
    @*/
  Object[] toArray(Object[] a) throws NullPointerException, ArrayStoreException;
```

Example Java I/O Library Specification (abridged)

public int **read**(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer. This method blocks until input data is available, end of file is detected, or an exception is thrown.
- If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.
- The first byte read is stored into element b[off], the next one into b[off+1], and so on. The number of bytes read is, at most, equal to len. Let *k* be the number of bytes actually read; these bytes will be stored in elements b[off] throughb[off+*k*-1], leaving elements b[off+*k*] through b[off+len-1] unaffected.
- In every case, elements b[0] through b[off] and elements b[off+len] through b[b.length-1] are unaffected.

- **Throws:**
  - IOException - If the first byte cannot be read for any reason other than end of file, or if the input stream has been closed, or if some other I/O error occurs.
  - NullPointerException - If b is null.
  - IndexOutOfBoundsException - If off is negative, len is negative, or len is greater than b.length - off

Example Java I/O Library Specification (abridged)

public int **read**(byte[] b, int off, int len) throws IOException

- Reads up to len bytes of data fro
bytes. An attempt is made to rea
number may be read. The numbe
an integer. This method blocks u
detected, or an exception is throw
- If len is zero, then no bytes are r
is an attempt to read at least one
the stream is at end of file, the va
one byte is read and stored into b
- The first byte read is stored into e
into b[off+1], and so on. The num
to len. Let *k* be the number of byt
stored in elements b[off] through
elements b[off+*k*] through b[off+
- In every case, elements b[0] thro
elements b[off+len] through b[b.length-1] are unaffected.

- **Throws:**
  - IOException - If the first byte can
end of file, or if the input stream
error occurs.
  - NullPointerException - If b is null.
  - IndexOutOfBoundsException - If
greater than b.length - off

- Specification of return
- Timing behavior (blocks)
- Case-by-case spec
  - len=0 ➔ return 0
  - len>0 && eof ➔ return -1
  - len>0 && !eof ➔return >0
- Exactly where the data is stored
- What parts of the array are **not** affected

- Multiple error cases, each with a precondition
- Includes "runtime exceptions" not in throws clause

Quality Attribute Specifications: Discussion

- How would you specify…
  - Availability?
  - Modifiability?
  - Performance?
  - Security?
  - Usability?

Testing – The Big Questions

1. **What is testing?**
   - And why do we test?
2. **To what standard do we test?**
   - Specification of behavior and quality attributes
3. **How do we select a set of good tests?**
   - Functional (black-box) testing
   - Structural (white-box) testing
4. **How do we assess our test suites?**
   - Coverage, Mutation, Capture/Recapture…
5. **What are effective testing practices?**
   - Levels of structure: unit, integration, system…
   - Design for testing
   - How does testing integrate into lifecycle and metrics?
6. **What are the limits of testing?**
   - What are complementary approaches?
     - *Inspections*
     - *Static and dynamic analysis*

Test Coverage

- Analysis: *the **systematic** examination of a software artifact to determine its properties*
  - We cannot test all inputs, so how can we be systematic?

- **Black box** testing
  - Systematically test different parts of the input domain
    - "domain coverage"
  - Test through the public API
    - focuses attention on client-visible behavior
  - No visibility into the code internals – a "black" box

- **White box** testing
  - Systematically test different elements in the code
    - "code coverage"
  - Can test internal elements directly – a "white" box
    - Good for quality attributes like robustness
  - Takes advantage of design information and code structure – a "white" box
    - "glass box" may be better terminology

Test Coverage

- Analysis: *the **systematic** examination of a software artifact to determine its properties*
  - We cannot test all inputs, so how can we be systematic?

- **Black box** testing
  - Systematically test different parts of the input domain
    - "domain coverage"
  - Test through the public API
    - focuses attention on client-visible behavior
  - No visibility into the code internals – a "black" box

- **White box** testing
  - Systematically test different elements in the code
    - "code coverage"
  - Can test internal elements directly – a "white" box
    - Good for quality attributes like robustness
  - Takes advantage of design information and code structure – a "white" box
    - "glass box" may be better terminology

Black Box: Equivalence Class / Partition Testing

- Equivalence classes
  - A partition of a set
    - Usually the input domain of the program
  - Based on some equivalence relation
    - Intuition: all inputs in an equivalence class will fail or succeed in the same way

Finding Equivalence Classes

- Cases in the specification
  - You may not have a spec – but still helps to think in this way
  - Remember that the spec may not be complete!
- One class per code path
  - Really white-box testing – boundary is fuzzy
  - Even with black box it can be useful to guess at the code structure
  - You may have an abstract algorithm to use
- Risk-based
  - Consider a possible error as a risk
  - Given an error, what classes of input could produce that error?
- Guideline – avoid writing many similar test cases
  - Suggests they are all from the same equivalence class

Equivalence Class Hueristics

- Invalid inputs
- Ranges of numbers
- Membership in a group
- Equivalent outputs
  - Can you force the program to output an invalid or overflow value?
- Error messages
- Equivalent operating environments

Boundary Value Testing

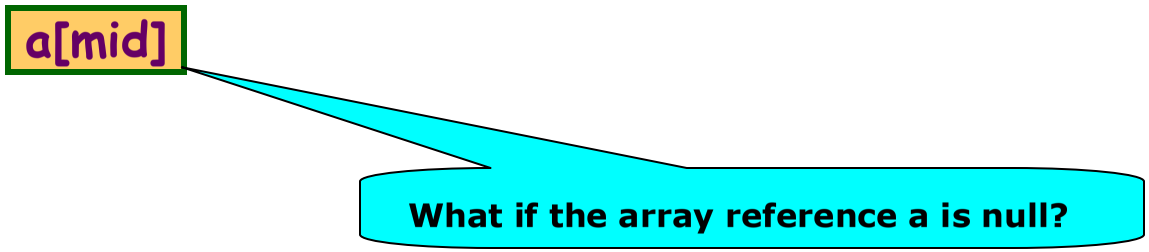
- What test case to choose from an equivalence class?
  - Which is most useful, i.e. likely to fail?
- Heuristic – boundary values
  - Extreme or unique cases at or around "boundaries"
    - Boundary of precondition – black box
    - Boundary of program decision point – white box
    - *Examples*: zero-length inputs, very long inputs, null references, etc.
  - Will usually find errors that are present in any other member of the equivalence class, but may find off-by-one errors as well

Combination Testing

- Some errors might be triggered only if two or more variables are at boundary values
- Test combinations of boundary values
  - Combinations of valid input
  - One invalid input at a time
    - In many cases no added value for multiple invalid inputs
- Subtlety required
  - What are the boundary cases for an application that deals with months and days?

Robustness Testing

- *Test erroneous inputs and boundary cases*
  - Assess consequences of misuse or other failure to achieve preconditions
  - Bad use of API
  - Bad program input data
  - Bad files (e.g., corrupted) and bad communication connections
  - Buffer overflow (security exploit) is a robustness failure
    - Triggered by deliberate misuse of an interface.
- Test apparatus needs to be able to catch and recover from crashes and other hard errors
  - Sometimes multiple inputs need to be at/beyond boundaries
- The question of responsibility
  - Is there external assurance that preconditions will be respected?
  - *This is a design commitment that must be considered explicitly*



Equivalence, Boundary and Robustness Example

- Program Specification
  - Given numbers a, b, and c, return the roots of the quadratic polynomial $ax^2 + bx + c$. Recall that the roots of a quadratic equation are given by:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Equivalence classes?

- Robustness test cases?