

# CMSC 433 – Programming Language Technologies and Paradigms Spring 2004

Testing and Specifications  
February 5, 2004

Some slides adapted from FSE'98 Tutorial by Michal Young and  
Mauro Pezze'

## Testing

- Execute program on sample input data
  - Check if output correct (acceptable)
- Goals
  - Increase confidence program works correctly
    - Acceptance Testing
  - Find bugs in program
    - Debug Testing

2

## Example (Black Box)

```
% java TestServlet HelloWorld /FooBar/Test > out
HTTP/1.0 200
Content-Type: text/plain
Hello /FooBar/Test
% diff out expectedOutput
```

3

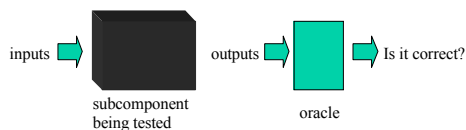
## Limitations of Testing

- Program runs on (very small) *subset* of input data
  - Exhaustive testing usually impossible
    - Too large input space (possibly infinite)
- Many situations hard to test
  - Parallel code (due to non-determinism)
  - Hard-to-reach states (e.g., error states)
  - Inadequate test environment (e.g., lack of hardware)
- Testing cannot prove absence of bugs
  - Especially a problem in security

4

## Black Box Testing

- Pick subcomponent of program
  - Internals of component not considered
- Give it inputs
- Compare against expected outputs



5

## Black Box Testing

- Pick subcomponent of program
  - Internals of component not considered
- Give it inputs
- Compare against expected outputs

– But how do I know what the expected outputs are?  
– Depends on the software specification ...

6

## Software Specifications

- A specification defines the *behavior* of an abstraction
- This is the *contract* between user and provider
  - Provider's code must implement the specification
  - Providers are free to change the implementation
    - So long as the new code still meets the specification
  - Users that depend implementation could be in trouble
    - Only rely on specification
- Black box testing essentially checks compliance of an implementation with its specification

7

## Good Specifications are Hard and Rare

- Very difficult to get people to write specifications
  - Even harder to keep them up to date
- Having specifications in a separate document from code almost guarantees failure
  - Rationale for **Javadoc**: extract a standalone specification from the code and embedded comments
- Hard to accurately and formally capture all properties of interest
  - Always finding important details not specified

8

## Specifications Help You Write Code

- Lots of subtle algorithms and data structures
  - Internal specs/invariants vital to correct implementation
- Example: Binary Search Tree
  - All nodes reachable from left child have smaller key than current node
  - All nodes reachable from right child have larger key than current node

9

## Specifications Help You Maintain Code

- In the real world, much coding effort goes into modifying previously written code
  - Often originally written by somebody else
  - Perhaps six different people have modified this code
- Documenting and respecting key internal specifications are the way to avoid a mess
- Documenting and respecting key external specifications are the way to avoid having your customers storm the office with torches and pitchforks

10

## Formal vs. Informal Specifications

```
static int find(int[] d,int x)
```

- An informal specification
  - If the array **d** is sorted, and some element of the array **d** is equal to **x**, then find() returns the index of **x** .....
- A formal specification
  - (for all  $i, 0 < i < d.length, d[i-1] < d[i]$  and there exists  $j, 0 \leq j < d.length, such that  $d[j] = x$ ) implies  $find(d,x) = j$  .....$

11

## Advantages and Disadvantages

- Formal specifications
  - Forces you to be very clear
  - Automated tools can check some specifications
    - Either at compile-time (static checking) or run-time (dynamic checking)
- Informal specifications
  - Some important properties are hard to express formally
    - Sometimes just difficult
    - Sometimes don't have the necessary formal notation
  - Some people are intimidated by formal specs

12

## Types of External Specifications

- Specifications on methods
  - Pre-conditions/requires: What must be true before call
  - Post-conditions/effects: What must be true after call
    - Often relates final values to initial values

```
// precondition: the array d is sorted
// postcondition:
//   returnValue >= 0 && d[returnValue] == x
//   or (returnValue == -1 && x does not occur in d)
static int find(int d[], int x);
```

13

## Types of Internal Specifications

- Specifications appearing within code itself
  - i.e., comments
- Loop invariants: condition that must hold at the beginning of each iteration of a loop
  - `d[0..i]` is sorted
- Data structure or field invariants
  - `elementCount <= elementData.length`

14

## Behavior vs. Function

- Side effects
  - Writes output to a file
  - Could block on a condition or mutex
- Performance
  - Should you specify performance of operations?
  - As hard as 451: what kind of bound (upper bound, amortized bound, expected bound, ...), order of bound, ...
  - But need at least informal specs
    - Random access is fast, insertion/deletion may be slow

15

## Specifications and Subtyping

- Liskov substitution principle (original? formal stmt)
  - If for each object *o1* of type **S** there is an object *o2* of type **T** such that for all programs **P** defined in terms of **T**, the behavior of **P** is unchanged when *o1* is substituted for *o2* then **S** is a subtype of **T**.
  - I.e, if anyone expecting a **T** can be given an **S**, then **S** is a subtype of **T**.
- If we *override* a method, how do the specifications of the original and new method relate?

16

## Specifications and Subtyping (cont'd)

```
// precondition: the array d is sorted
// postcondition:
//   returnValue >= 0 && d[returnValue] == x
//   or (returnValue == -1 && x does not occur in d)
static int find(int d[], int x);
```

- If we override this method, can the new method
  - Have true as a precondition?
  - Have precondition “**d** is sorted and exists *i* s.t. `d[i] == x`”?
  - Have postcondition “`returnValue == -1` or `returnValue` is first index such that `d[returnValue] == x`”?
  - Throw `NoSuchElementException` rather than returning `-1` when `x` does not occur in `d`?

17

## What Makes a Good Specification?

- Sufficiently restrictive
  - Forbids unacceptable implementations
- Sufficiently general
  - Allows all acceptable implementations
- Clear
  - Easy to understand
  - A little redundancy may help (some people disagree)

18