Getting those Bugs Out --Software Testing

1. Issues in software testing and reliability
2. Test sets, test selection criteria, and ideal test sets.
3. Defect Testing
   3.1 Black Box Testing
   3.2 White Box Testing
4. System Test, Acceptance Test

2

Difficulties of testing software

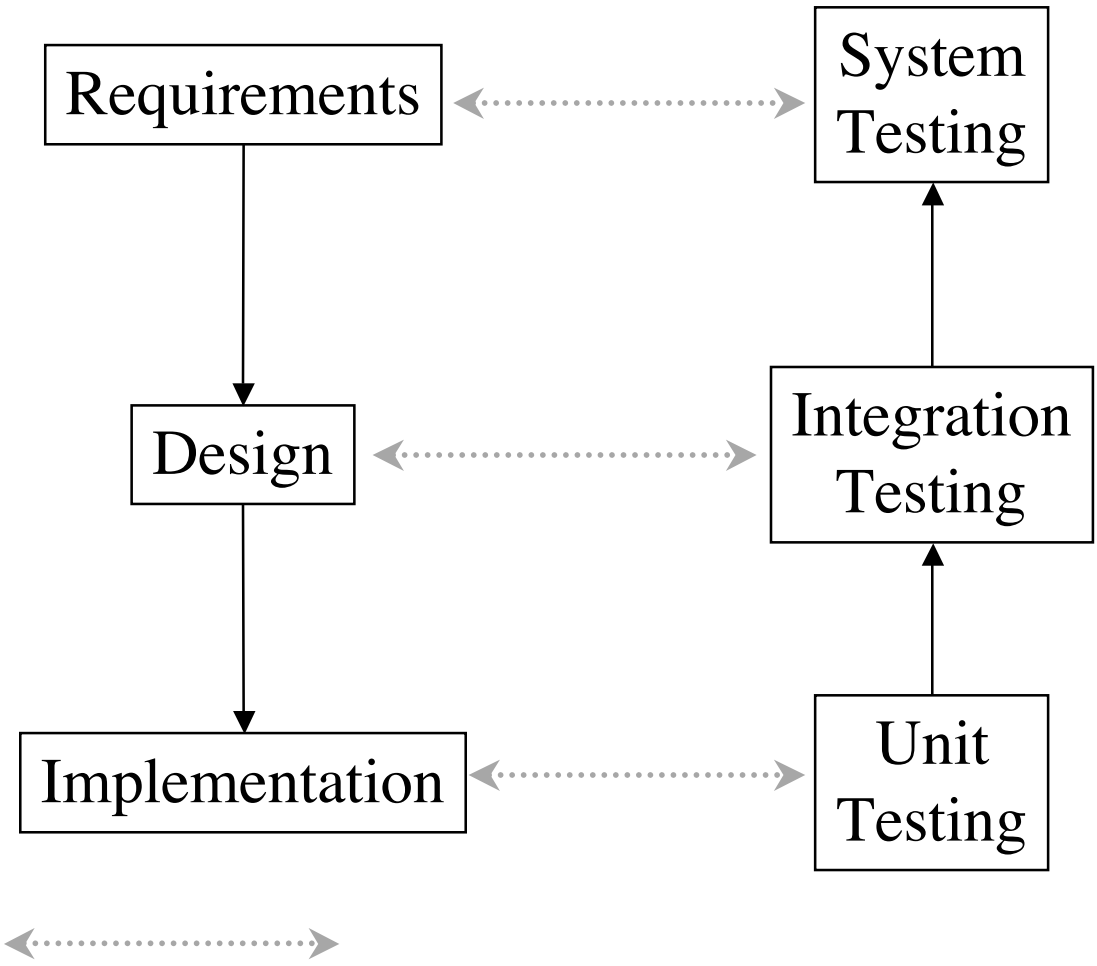
- *Input Domain Size* The input space is enormous (typically infinite).
- *Fish in the sea problem*. We can never tell if faults remain in the system, only what faults have been found.
- *Software Complexity* Software has numerous internal states and paths (too numerous to test them all).
- *High reliability requirements* Very hard to test systems that have extremely stringent quality requirements.
- *Time constraints* When do you stop testing?

PROBLEM: how do you pick test sets???

- *Requirements Clarity* Sometimes it can be quite difficult to tell whether or not the system is performing correctly.
- *Variable User Perception* Users have different ideas of what they want the software to do.

2

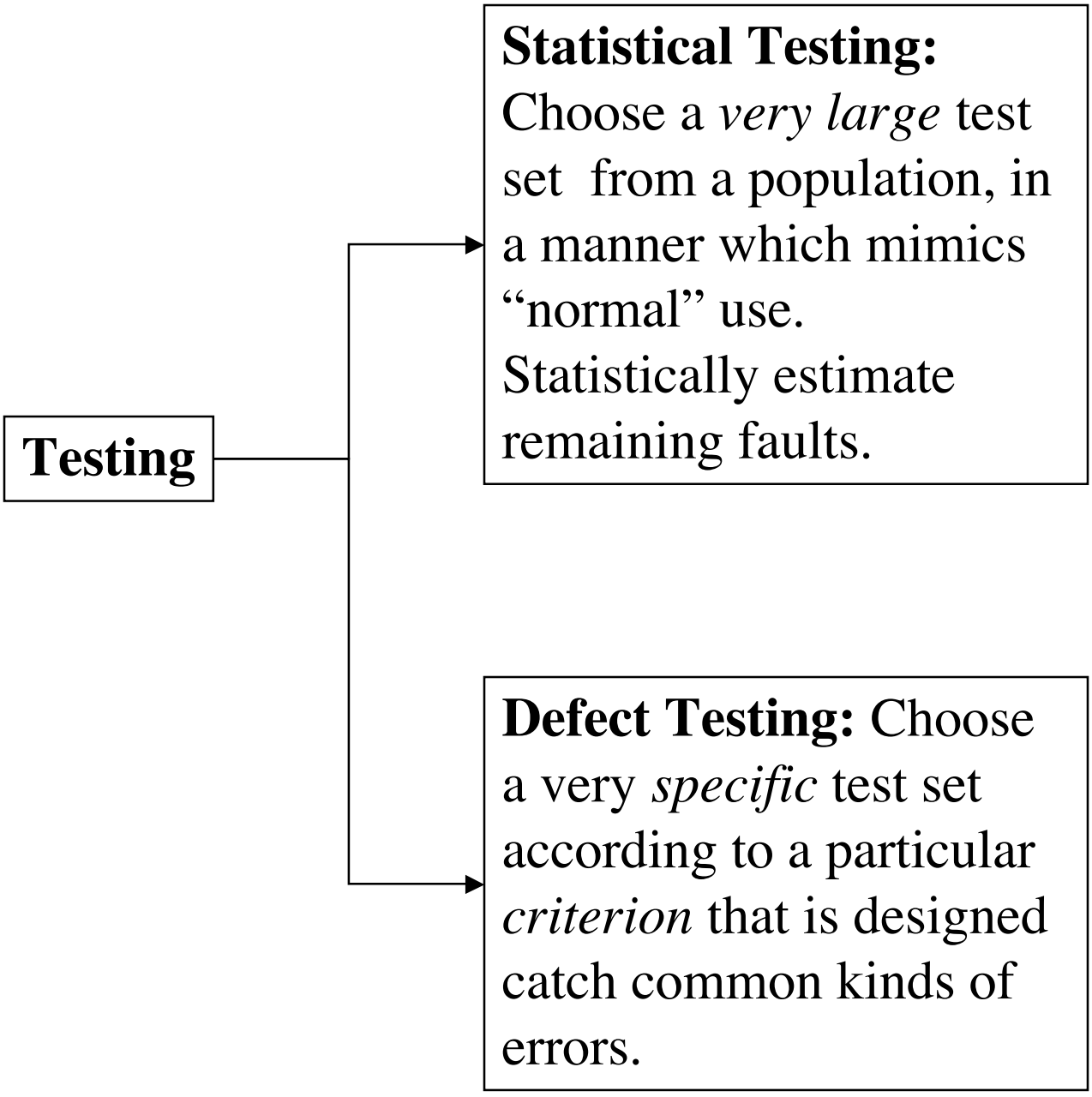
Recap of Lifecycle Testing



Testing relationship. Can mean 2 things:
1. Faults introduced in this step are detected by the corresponding testing phase.
2. Checks to make sure that the product conforms to the corresponding lifecycle document.

This lecture: Mostly focused on Unit testing, although the same principles apply to other levels of testing.

Two Main kinds of testing.



A Cautionary Tale

/*returns x^(3/2) */
```
float exp3over2pos(float inp) {/*Version 0.1 */
   return (inp*inp/sqrt(inp));
};
```
Test Inputs: 3, 4, -5!!!
```
float exp3over2pos(float inp) {/*Version 0.2 */
   if (inp < 0)
         return(inp*inp/sqrt(-inp));
   else return (inp*inp/sqrt(-inp));
}
```
Test Inputs: -5, -3, -2
```
float exp3over2pos(float inp) {/*Version 0.3 */
   if (0 == inp)
        throw ZeroDivideException;
   else if (inp < 0)
        return(inp*inp/sqrt(-inp));
        else return (inp*inp/sqrt(inp));
};
```

Test Inputs: 0

"Hello, System test: This function returns |x|^(3/2) for float x."

6

Morals of this story:

- *Regression Testing:* after making any changes to the program, test on every input that **previously worked** with the the old version of the program.

- *Picking test sets is HARD!*

  - *Black Box testing:* Using the specification as a guide, identify different categories of inputs, and select one or more inputs from each of those categories.
  - *White box testing:* Using the structure of the program as a guide, identify categories of inputs that exercise different parts of a program's structure, and select one or more inputs from each category.

- *No testing method is guaranteed to find faults, failures, errors etc.*

6

Some Terminology

- *Failure:* Incorrect system behaviour--not what the customer wanted.
- *Fault:* the part of the program that caused this failure to happen.
- *Input Domain:* The set of all possible inputs to the program *P*, denoted by *D*.
- *Test Set:* a set *T* such that: $T \in 2^D$
- *Test selection criterion TC:* $TC \subseteq 2^D$
- *T satisfies TC if:* $T \in TC$
- *Ideal T:* one that always reveals a fault.
- TC1 *finer than* TC2 if for every program *P*, every set *T* that is in TC1 there is a set *t* in TC2 such that

$$t \subseteq T$$

Functional or Black Box Testing

Test selection Criterion: the program's specification (irrespective of implementation)

Specification of a binary search routine:

X is a sorted array of keys of size n > 1

k is an input key.

If X has the key, return smallest j such that X[j] = k; else throw a "notFound" exception.

How to identify input categories based on the specification.

1. Inputs which conform to the pre-conditions.
2. Inputs which don't.
3. Inputs which give rise to each type of answer.

How to do this for binary search?

What are the benefits of this approach?

-Test creating process carefully looks at specification.
-Testing "true" to specifications
-Can be done by someone unfamiliar with the implementation of the system

What are the disadvantages of the approach?

-May not test for faults due to internal implementation problems.
-May not test combinations of categories.
-Usually informal, hard to verify *satisfaction of criterion.*

White Box Testing

Test selection Criterion: the program's <u>implementation (irrespective of specification)</u> but based on some abstraction/approximation.

1. *Statement Coverage Criterion:* a test set T, such that by executing program P on T, every statement in the program P is executed.
2. *Test Coverage measure:* for any test T *any* criterion, how good is it (% of coverage).

```
float myfun(float x) { /*I don't know what this program does */
   float y;
   if (x == 0)
        throw xZeroEx;
   else if (x >= 1)
        throw xOneEx;
   if(x < 0) y= -x;
   return(y/sqrt(x*(1-x)));
}
```

Some Sample Test Sets:

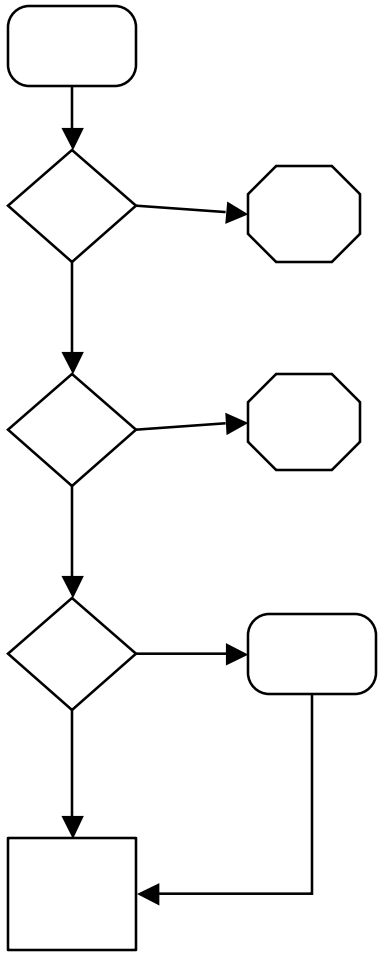
```
{1,-2} {0,-2}, {0,-2, 3}, {0, -1, 2}
```

Do they satisfy criterion? If not, what's the coverage?

More Whitebox Testing

Edge coverage criterion: every edge in a program's control flow graph is covered:

```
float myfun(float x) {
   float y;
   if (x == 0)
        throw xZeroEx;
   else if (x >= 1)
        throw xOneEx;
   if(x < 0) y= -x;
   return(y/sqrt(x*(1-x)));
}
```



Some Sample Test Sets:

```
{1,-2} {0,-2}, {0,-2, 3},
{0, -1, 2}
{0, -1, 2, -2}
{0, -2, 3, 0.8}
{0,-1,2,0.7}
```

Edge coverage criterion is <u>finer</u> than *statement coverage criterion*

11

Proof that edge coverage is finer that statement coverage.

This means that for every program, every test case that gets E. cvg also gets S. cvg.

Suppose not.

Then there is some program P, for which a test set T achieves edge coverage, but fails to get S coverage, i.e., some statement S in P is not executed by T.

But let's assume that P doesn't have dead Code.

Then S must be reachable by some edge E.

But then T must execute E (since T gets edge coverage).

So S must be executable.

So T must execute S. Reductio ad Absurdum.

<u>"For programs without dead code, edge coverage is finer than statement coverage"</u>

11

12

Status of White box testing

- *Non-ideal:* test sets that satisfy white box coverage criteria are not guaranteed to find faults.
- *Can be automated:* various tools are available which measure the level of test coverage.
- *Well studied:* Several different criteria, of varying "fineness" have been proposed, and the comparisons are well sorted out.
- *Widely adopted:* Many organizations use white box coverage measures to gauge testing effective.
- *Experimentally borne out:*
  Ostrand et al, Malaiya et al: 85% -95% edge coverage leads to significantly lower faults in the field.

12