Developer Testing
Theory and Concept

Introduction

Developer Testing:

- Front line testing.
- One of the most important tasks a developer can do.
- Must be comprehensive!

Introduction - Fundamental Counting Principle

The **fundamental counting principle** states that if there are p ways to do one thing, and q ways to do another thing, then there are $p \times q$ ways to do both things.

Example 1:

Suppose you have 3 shirts (call them A , B , and C), and 4 pairs of pants (call them w , x , y , and z). Then you have

$$3 \times 4 = 12$$

possible outfits:

$$\mathrm{A}w, \mathrm{A}x, \mathrm{A}y, \mathrm{A}z$$
$$\mathrm{B}w, \mathrm{B}x, \mathrm{B}y, \mathrm{B}z$$
$$\mathrm{C}w, \mathrm{C}x, \mathrm{C}y, \mathrm{C}z$$

Introduction - Fundamental Counting Principle

Example 2:

Suppose you roll a 6 -sided die and draw a card from a deck of 52 cards. There are 6 possible outcomes on the die, and 52 possible outcomes from the deck of cards. So, there are a total of

$$6 \times 52 = 312$$

possible outcomes of the experiment.

The counting principle can be extended to situations where you have more than 2 choices. For instance, if there are p ways to do one thing, q ways to a second thing, and r ways to do a third thing, then there are $p \times q \times r$ ways to do all three things.

Example - The Stack

```
#define STACKSIZE 10

class Stack {
public:
    Stack();  // constructor
    ~Stack();  // destructor
    int pop();
    int peek();
    bool push(int);
    bool isEmpty();
private:
    int top;
    int stack[STACKSIZE];
};
```

Example - The Stack States & Operations

- Count the number of possible states.
  - overflow
  - underflow
  - neither underflow nor overflow
- Count the number of possible operations.
  - push
  - pop
  - peek
  - isEmpty
- Use the fundamental counting principle to determine the *minimum* number of tests.
  3 X 4 = 12.

Example - The Stack Tests

- Place the stack in underflow.
  - Execute, push, pop, peek, isEmpty
  - Make sure to do all operations multiple times in this state.
- Place the stack in overflow.
  - Execute, push, pop, peek, isEmpty
  - Make sure to do all operations multiple times in this state.
- Place the stack in neither overflow nor underflow.
  - Execute, push, pop, peek, isEmpty
  - Make sure to do all operations multiple times in this state.

Example—Make Tests Automated

- Never use user input!
- Use defines and constants based on the object / function / system being tested.

```
// testing filling the stack and overflow
cout << "Filling stack..." << endl;
for (int i = 0; i < STACKSIZE*MULTIPLIER; i++) {
    value = randome_int();
    if (stack.push(value)) {
        cout << "pushed: " << value << endl;
    } else {
        cout << "overflow error: " << value << " not pushed" << endl;
    }
}
cout << endl;
```

Example—Use Random Operations

- Test your object / function / system 10,000s or 100,000s of times!
- Use random operations.

```
int choice=0;
for (int i = 0; i < STACKSIZE*RMULTIPLIER; i++) {
    choice = rand() % CHOICES + 1;
    switch (choice) {
        case 1:
        case 2:
            // push here
            break;
        case 3:
        case 4:
            // pop here
            break;
        case 5:
            // peek here
            break;
        case 6:
            // isempty here
            break;
    }
}
```

Conclusion

- Use the Fundamental Counting Principle to guide your testing.
- Use these concepts as the basis for your testing framework (unit testing, test driven development, etc.)