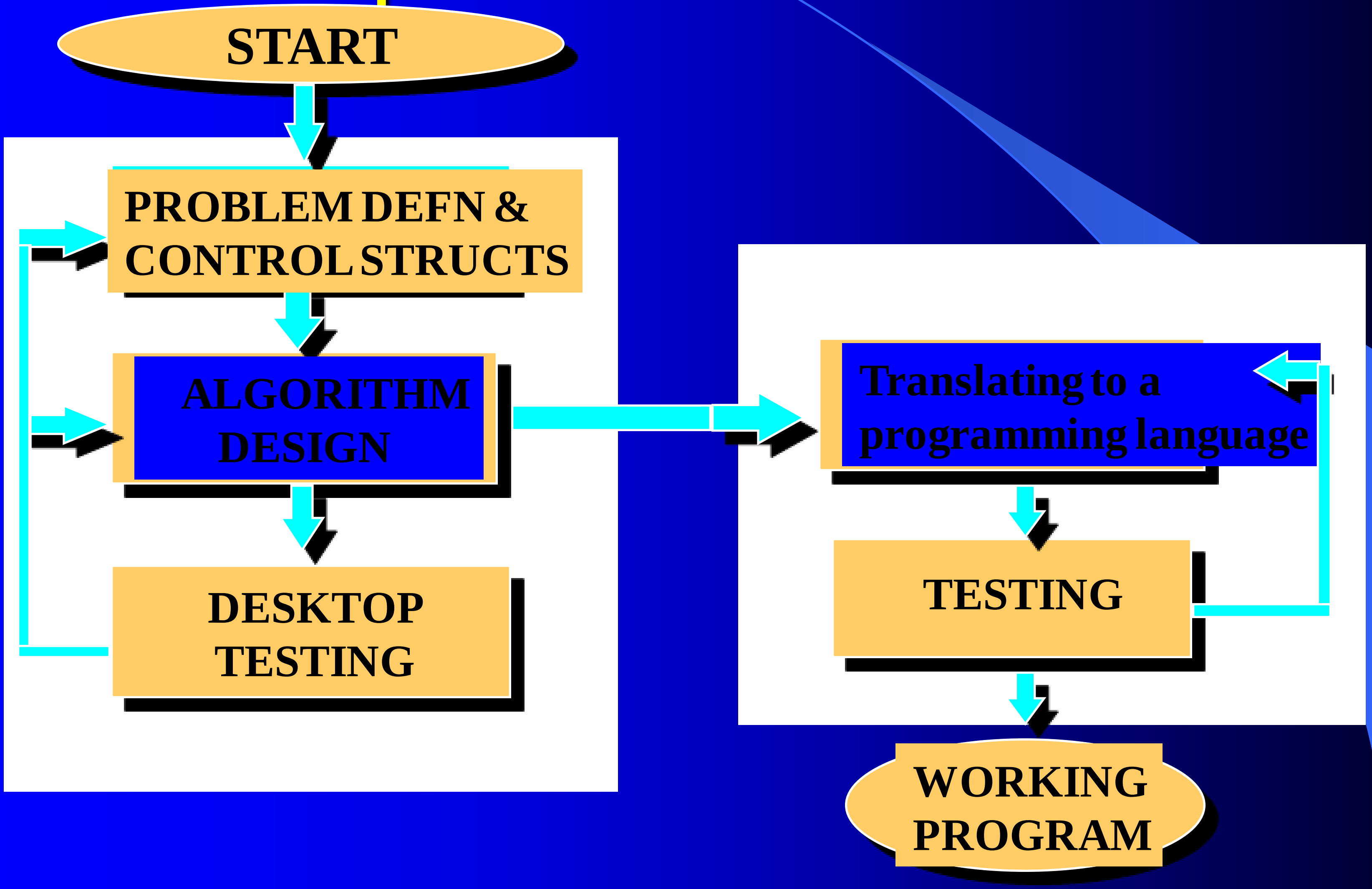
OOJ Lecture 6
Software Engineering: Modularity & Information Hiding

Review: Program Development Process


START
PROBLEM DEFN & CONTROL STRUCTS
ALGORITHM DESIGN
DESKTOP TESTING
Translating to a programming language
TESTING
WORKING PROGRAM

- Self test questions:
  - What is the goal of program modularity?
  - What is information hiding?
  - What are the purposes of creating a function?

The Goal of Program Modularity

- To make each function (or module) in the program like a black box, such that for any specific input, you can accurately predict the corresponding output

Information Hiding

- The above notion of black box is what may be termed information hiding, where the private/inside information of a function, eg the format of a file or the way data structure is implemented, is encapsulated.

Exercise (Information Hiding)

```
A bad example
============
double calculate_amount_owing(CustObject
  customer)
{
   if (customer.water_usage <= 100)
          customer.amount_owing =
               customer.water_usage * 0.65;
   else
     customer.amount_owing = 100 * 0.65 +
               customer.water_usage * 0.65;
   return customer.amount_owing
}
```

Exercise (Information Hiding)

A good example

```
// This function computes and returns the amount owing
// based on the usage and TARIFF_LOWER, TARIFF_UPPER and
// CUTOFF_VOLUME.
double calculate_amount_owing( const int usage
{
   final int CUTOFF_VOLUME = 100; // Declare constants
   final double TARIFF_LOWER = 0.65; // within the function itself.
   final double TARIFF_UPPER  = 0.70;
   double amnt_owing;
   if (usage <= CUTOFF_VOLUME)
       amnt_owing = usage * TARIFF_LOWER;
   else
       amnt_owing = (CUTOFF_VOLUME * TARIFF_LOWER) +
          (usage - CUTOFF_VOLUME) * TARIFF_UPPER;
   return amnt_owing;
}
```

Eight purposes of creating a function

Purpose #1: To reduce complexity

- keeping one specific task per function
- making central points of control
- making a section of code readable

Purpose #2: To avoid duplicate code

- similar code in a program implies an error in decomposition

Purpose #3:
To limit effects of changes

- areas likely to change include H/W dependencies, input/output, complex data structures and business rules and charges
- facilitating modifications
- improving portability

Purpose #4:
To hide data structures

- avoiding the messy details of manipulating data structures, eg reference operations (push, pop, delete_a_node, add_a_node, create_a_node)

Purpose #5:
To hide global data

- through parameter passing

Purpose #6:
To promote code reuse

- saving time and money

Purpose #7:
To isolate complex operations

- complex operations are error-prone
- by isolating complex operations, such errors could be contained

Purpose #8: Simplify complicated boolean tests

- making code more readable
- Not so good:

  ```
  if (x > 0 && y > 0 && z > 0 && w > 0)
  ```
- Better:

  ```
        if (numbers_positive(  x, y, z, w ))
  ```

Example of a clumsy function: Identify the pitfalls

```
// Bad example!!
/* This function updates the position (x,y) of a molecule,
    depending on the value returned by rand(). */
void update_position( )
{  int step;
  step = rand()%4 - 2;
  if (step >= 0)
     step = step + 1;
  if (abs(step) > 1)
  { int new_position;
    new_position = y + step/2;
    if (abs(new_position) <= 10)
       y = new_position;
  }
```

Example –cont’d

```
else
  {
    int new_position;
    new_position = x + step;
    if (abs(new_position) <= 10)
       x = new_position;
  }
System.out.println( " " current (x,y) position is "
    + “(“ + x + “,” + y +”)” );
}
```

Pitfalls:

- does not keep one specific task per function
- does not make central points of control
- does not avoid similar code
- does not hide global data using parameter passing

An improved version of the clumsy function

```
// This function updates the position (x,y) of a molecule, depending on
// the value returned by which_direction().
void update_position( int& x, int& y )
{  int step;
   step = which_direction();
  if (abs(step) > 1)       // If step is 2 (-2) a molecule moves 1 unit in
    move(y, step/2);       // the +y (-y) direction,
   else                    // If step is 1 (-1) a molecule moves 1 unit in
    move(x, step);         // the +x (-x) direction.
}
```

An improved version of the clumsy function

```
// This function updates the x(y)-coordinate of a molecule if
// the new position is not outside the container.
// Otherwise, the molecule remains against the container wall.
void move( int current_position, int unit_step )
{
  int new_position;

  new_position = current_position + unit_step;
 if (abs(new_position) <= 10)          //If the new position is within the
   current_position = new_position;    //container or on the container
                                       //wall, return the new x(y)-coord.
}
```

An improved version of the clumsy function

```
/* This function randomly returns one of the four integers,  -2, -1, 1, 2, which
     determine the movement of a molecule in one of the four allowable
     corresponding directions: +x, -x, +y, -y. */
int which_direction()
{  int n;
   n = rand()%4 - 2;
   if (n >= 0)
      n = n + 1;
  return n;
}
// This function displays the current molecule position.
void display_position (int x, int y)
{
    System.out.println( " Current molecule position is "
     + “(“ + x + “,” + y +”)” );    }}
```

Summary: Program modularity and Info Hiding

- Understand what program modularity and information hiding are
- Understand that program modularity can be achieved via functional (object) decomposition
- Understand the purposes of creating a function