

502045

# Software Engineering

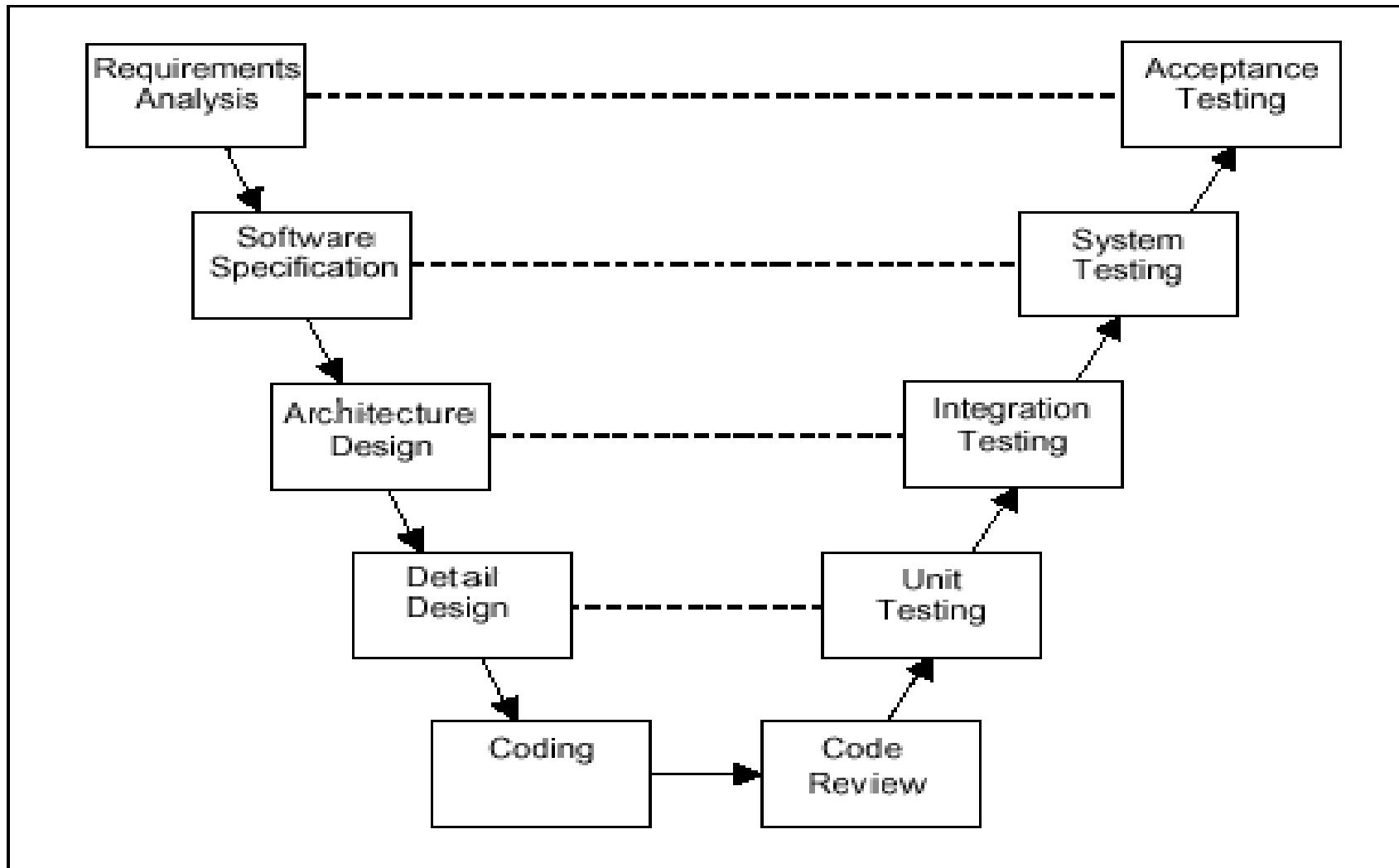
## Chapter 07

### Lesson 09: Code Quality

# Code Quality

- Agenda
  - Review Process
  - Common Defects

# Review Process - Where the Coding is?



# Review Process – The Quality Triangle

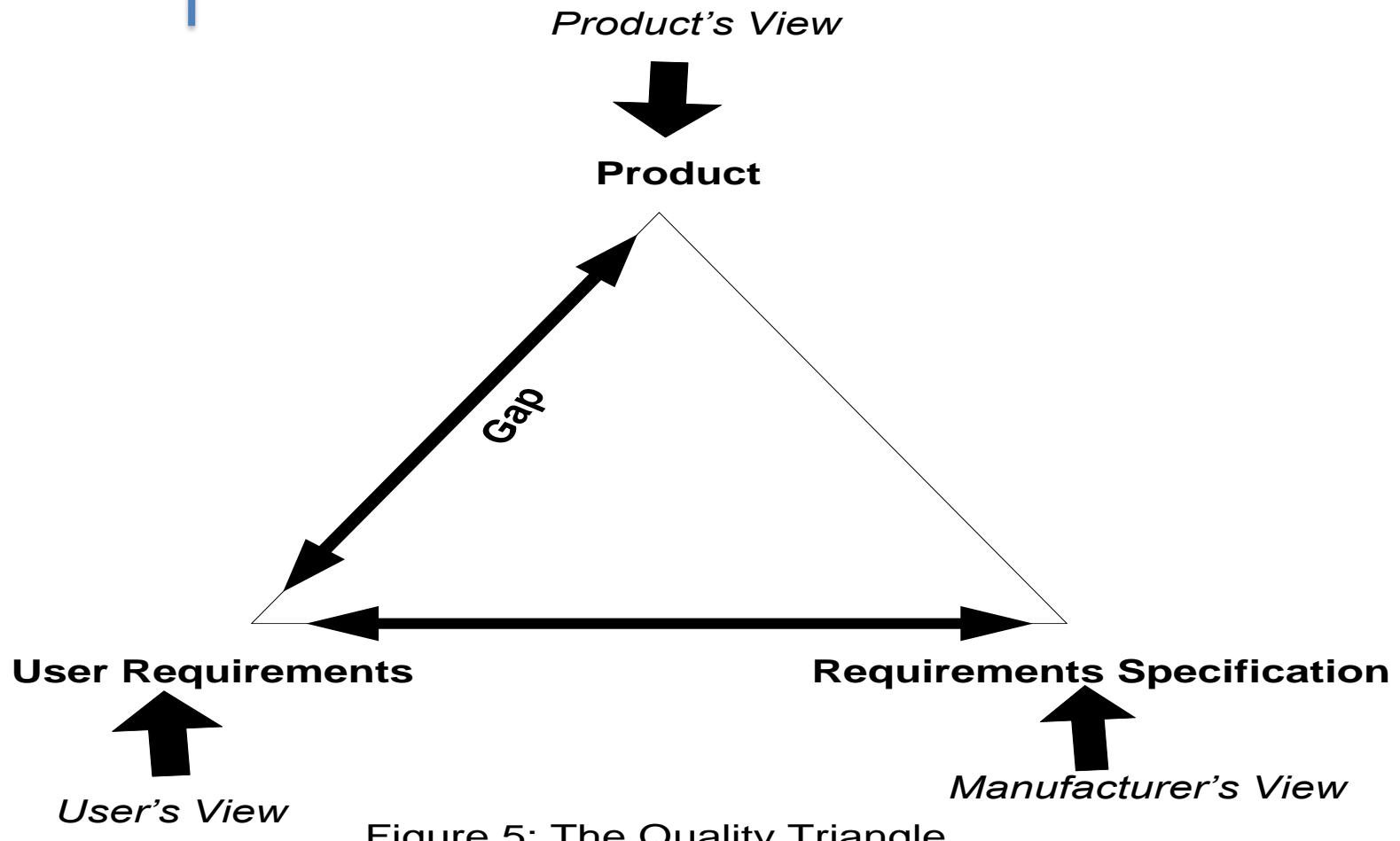


Figure 5: The Quality Triangle

# Review Process - The UR-RS gaps

- The gap is likely to include:
  - Wrong interpretation of requirement and ambiguity in the specification.
  - Requirements identified after development commenced [started].
  - Changes to specified requirements identified after development commenced
  - Requirement ignored by the developers because they were too difficult to implement.

# Review Process - The Software-UR Gap

- The gap occurs because the software doesn't satisfy the user requirements
- The size of the gap is directly dependent on the side of other 2 sides of triangle

# Review Process - Closing gaps between 3 views

- Static test:
  - Review: online/offline
  - Inspection
- Dynamic test:
  - Unit Test
  - Integration Test
  - System Test
  - Acceptance Test

# Common Defects & Practices

## Hard code constants

- Issue with giving a fixed value in codes, for example:  
`dgrView.PageSize = 10;`  
The problem occurs when you should change these values multiple times!!!
- Preventive Action: define constants in the common constant module or in a configure files

# Common Defects & Practices

## Array Index Start from 0

- Issue with below C-Language codes?

```
int i, a[10];
```

```
for (i=1; i<=10; i++) a[i] = 0;
```

# Common Defects & Practices

## The Dangling else Problem

- Issue with below C-Language codes?

```
if (x == 0)
    if (y == 0) error();
else {
    z = x + y;
    f (&z);
}
```

# Common Defects & Practices

## Null Pointer Exception

- Issue: the developer got Null-Pointer-Exception run-time error, while he/she did not detect that when compiling the codes

`pPointer->member = 1;`  
`strReturn = objDoc.SelectNodes(strName);`
- Cause: the developer does not check null or think about null object before accessing object's value.
- Preventive: Should check null before accessing object or pointer before using its member

`If ( pPointer != NULL ) pPointer->member = 1;`  
`If (objDoc != NULL)`  
`strReturn = objDoc.SelectNodes(strName);`

# Common Defects & Practices

## Detect Common Defects Sample

```
•     public bool IsValidLogin(string userName, string password)      {  
•         SqlConnection con = null;  
•         SqlCommand cmd = null;  
•         bool result = false;  
•         try {  
•             con = new SqlConnection(DB_CONNECTION);  
•             con.Open();  
•             string cmdtext = string.Format("SELECT * FROM [Users] WHERE [Account]='{0}' AND  
•                                         [Password]='{1}' ", userName,  
password);  
•             cmd = new SqlCommand(cmdtext);  
•             cmd.Connection = con;  
•             cmd.CommandType = CommandType.Text;  
•             result= cmd.ExecuteReader().HasRows;  
•             cmd.Dispose();  
•             con.Dispose();  
•             return result;  
•         }  
•         catch (SqlException) {  
•             return false;  
•         }  
•     }
```

# Common Defects & Practices

## Programming Practices 1

- Issue with variables or create objects in Loop?

```
for (int i=0; i<dt.Rows.Count-1; i++)  
{  
    string strName;  
    strName = dt.Rows[i]["Name"].ToString();  
    //do something here  
}
```

**Impact to the application performance!!!**

- Cause: memory is allocated repeatedly.

# Common Defects & Practices

## Programming Practices 2

- Code redundant issues:
  - Create new Object while we can reuse the object in previous command:

```
BeanXXX bean = new BeanXXX();  
bean = objectYYY.getBeanXXX();
```
  - Variables are declared in based class but it is not used

# Common Defects & Practices

## Programming Practices 3

- Avoid using an object to access a *static* variable or method. Use a class name instead.

```
classMethod();           //OK
AClass.classMethod();    //OK
anObject.classMethod();  //AVOID!
```

- Avoid assigning several variables to the same value in a single statement.

```
fooBar.fChar = barFoo.lchar = 'c'; // AVOID!
```

# Common Defects & Practices

## Programming Practices 4

- Do not use the assignment operator in a place  
`if (c++ = d++) { // AVOID!`

    ...  
    }

should be written as:

```
if ((c++ = d++) != 0) {  
    ...  
}
```

- Do not use embedded assignments in an attempt to improve run-time performance.

`d = (a = b + c) + r;`

# Common Defects & Practices

## Programming Practices 5

- File operations: file read operations must be restricted to a minimum
- Clear content of big structure after use: always clear() the content of Collection/Map objects after use
- Be economical when creating new objects
- In program language that has no garbage collector (i.e C, C++): free allocated memory after use:

```
{  
    double* A = malloc(sizeof(double)*M*N);  
    for(int i = 0; i < M*N; i++){  
        A[i] = i;  
    }  
}
```

**memory leak: forgot to  
call `free (A)` ;  
common problem in C,  
C++**

# Common Defects & Practices

## Programming Practices 6

- Use parentheses liberally in expressions involving mixed operators to avoid operator precedence problems
  - if (a == b && c == d) // AVOID!
  - if ((a == b) && (c == d)) // RIGHT
- Try to make the structure of your program match the intent [goal], for example:

```
if (booleanExpression) {  
    return true;  
} else {  
    return false;  
}
```

should instead be written as

```
return booleanExpression;
```

