Currennt Issues/Debt

- Debugging
- Reusabality of Exisisting or Third Party Code
- Skill Gap, Knowledge Transfer
- Cognitive Complexity is high
    - Names ,Comments
- Lack of Low Level Design Documents (Sequence Diagram ,Communication Diagram , Activity diagram)
- Source Code is the Single Source Truth
- Legacy Code
    - Lack of Testing
- Bugs

> *Code Coverage /Test Coverage 100% , Not Bug Free*

- Lack of Effective Testing
- Lack of Testable Code

How to improve Code Testablity - Unit testing

- Unit Testing ?
- unit : function
- testing : checking Behavior of the function

```
void add(int x,int y){
    int sum=0;
    sum=x+x;
   print(sum); // printf,cout, print,Console.WriteLine, System.Out.Println()
}

add(10,20)

Given 10 for x and 20 for Y when add invoked x+y =30 is expected
```

Testable ? : yes , System testing, e2e, UAT

Test Pyramid : Unit -> Integration -> System Testing - Sensible Measurement Points - Code is not tetsable if we can't measuer the results - Improve code Predicatability - Pure functions - Reduce Cyclomatic Complexity

How to add mesauring points in the code for Unit Testing

```
 - return value
 - global state
 - object state
 - mock state

 ```
```

int add(int x,int y){ int sum; sum=x+y; print(sum); // printf,cout, print,Console.WriteLine, System.Out.Println() return sum; } add(10,20) Given 10 for x and 20 for Y when add invoked x+y =30 is expected
```

```
//extern (visible to unit test env)
int sum;
void add(int x,int y){
     sum=x+y;
     print(sum); // printf,cout, print,Console.WriteLine, System.Out.Println()

 }
```

Code Improvements for Unit Testing

- SRP (One thing at a time , One reason to change) , Separate concerns
- Measuement Points
- Predictable , avoid side effects
- Reduce Cyclomatic Complexity
    - ideal value is :1
    - Accepatble Value is : 1 - 3
- Never write unit test code for your code
    - Pair Programming

Summary (Cyclomatic Complexity , Pure Functions and Seperation of Concerns)

- DRY (Do not repeat your self)
- Don't change public Api's (Impact Analysis)
- YAGNI
- Separate I/o operation from pure function
- Code for testability

```
int add_v1(int x,int y){
int sum=x+y;
std::cout<<sum<<std::endl;
return sum;
}
```

Behavior List

- Computation - addition and return result
- Has dependency on std::cout and interact with dependnecy (delegation)
  add_v1 Test Scenarios computation : Value Based Testing

```
      assert(add_v1(10,20),30);
```

Dependency Interaction : Behavior Testing or Interaction Testing

```
                  Substitute Dependency for testability
```

```
#include <stdio.h>
```

```
// Abstraction: function pointer for output int add(int x, int y, void (*output)(int))
{ int sum = x + y; output(sum); // depends on abstraction return sum; }

// Concrete output function void printToConsole(int value) { printf("%d\n", value); }

int main() { add(3, 4, printToConsole); // inject concrete behavior return 0; }
```

```
#include <stdio.h> #include <stdlib.h> #include <string.h>

// Function pointer type for output behavior typedef void (*OutputFunc)(int);

// Production function that uses injected output logic int add(int x, int y,
OutputFunc output) { int sum = x + y; output(sum); return sum; }

// Concrete output: print to console void print_to_console(int value) { printf("%d\n",
value); }

// Test collector: store value in buffer typedef struct { int* values; int count; }
OutputCollector;

void collect_output(int value) { extern OutputCollector collector;
collector.values[collector.count++] = value; }

// Global test output buffer OutputCollector collector = { NULL, 0 };

// Test function void run_test() { // Initialize buffer collector.values =
(int*)malloc(sizeof(int) * 10); // space for 10 values collector.count = 0;

// Call add with test output function
int result = add(5, 7, collect_output);

// Assertions
if (result == 12 && collector.count == 1 && collector.values[0] == 12) {
    printf("✅ Test passed\n");
} else {
    printf("❌ Test failed\n");
}

free(collector.values);

}

int main() { // Use real console output add(3, 4, print_to_console);

// Run the test
run_test();

return 0;

}
```