SDM College of Engineering and Technology

Dhavalagiri, Dharwad-580 002. Karnataka State. India.
Email: principal@sdmcet.ac.in, cse.sdmcet@gmail.com
Ph: 0836-2447465/ 2448327 Fax: 0836-2464638 Website: sdmcet.ac.in

DEPARTMENT
OF
COMPUTER SCIENCE AND ENGINEERING

MINOR WORK REPORT

[22UHUC5000- Software Engineering and Project Management]

Odd Semester: September 2024-January 2025

Course Teacher: Dr. U.P.Kulkarni

2024- 2025

Submitted by
By

Srushti A Pattar
2SD22CS108
5th Semester B division

Table Of Contents:

About Software Engineering:

Software engineering is the discipline that focuses on designing, developing, maintaining, and managing software systems in a systematic, efficient, and reliable way. It combines principles from computer science, engineering, and project management to create software solutions that meet user requirements and are scalable, maintainable, and high-performing.

Software engineering involves the systematic application of engineering principles to the development, operation, and maintenance of software. It encompasses several key areas:

- **Requirements Engineering**: Gathering and specifying what the software needs to do.
- **Design**: Creating architecture and detailed design for software that meets the requirements
- **Development**: Writing code using programming languages, frameworks, and tools
- **Testing**: Ensuring the software works as intended, through unit, integration, system, and acceptance testing.
- **Maintenance**: Modifying and improving software after it has been deployed.

ASSIGNMENT 1:

Problem Statement: Write C program to show that c programming language supports only call by value

Theory:

In C programming, **Call by Value** means that when a function is called, the value of the arguments is passed, not the actual variable. Any modification made to the parameter inside the function will not affect the actual value of the variable outside the function. C does not support **Call by Reference** natively (i.e., it doesn't directly allow passing the address or reference of a variable, though we can simulate it using pointers).

In **Call by Value**, the function creates a copy of the arguments, and the function operates on that copy. As a result, any changes made to the function parameter do not reflect in the actual variable used for the function call.

Design:

- Create a simple C program with a function that tries to modify the value of a variable.
- Inside the function, we'll modify the local copy of the parameter.
- Outside the function, we will print the original variable to show it is unchanged.

Program:

```
#include <stdio.h>

// Function definition using call by value

void modifyValue(int x) {

printf("Inside function, before modification: x = %d\n", x);
x = 20;  // This changes only the copy of x, not the original
printf("Inside function, after modification: x = %d\n", x);
}

int main() {
int a = 10;
```

```
// Print the value before function call
printf("Before function call: a = %d\n", a);

// Function call
modifyValue(a);

// Print the value after function call to show it remains unchanged
printf("After function call: a = %d\n", a);

return 0;
}
```

Sample input and output:

Before function call: a = 10
Inside function, before modification: x = 10
Inside function, after modification: x = 20
After function call: a = 10

References:

1) Software Engineering: A Practitioner's Approach by Roger S. Pressman (9th edition)
2) C Programming Language by Brian Kernighan and Dennis Ritchie

ASSIGNMENT 2:

Problem Statement: Study the concept "Usability", and prepare a report on "Usability" of at least two UI's of major software products you have seen.

Theory:

Usability refers to the ease with which a user can interact with a system or product to achieve a specific goal efficiently and effectively. The concept of usability is a critical aspect of user experience (UX). In this report, I will assess the usability of two major software products: **Google Docs** and **Microsoft Outlook**.

Usability Analysis of Google Docs

Overview: Google Docs is a cloud-based word processing tool that allows multiple users to create, edit, and collaborate on documents in real-time.

Usability Factors**:**

- ***Learnability****:* Google Docs has an intuitive and clean interface that makes it easy for new users to start using the tool. The menus and options are similar to traditional word processing software (such as Microsoft Word), making it easy for users familiar with similar tools.
- ***Efficiency****:* Once users get familiar with the layout and features, they can perform tasks quickly. Features like real-time collaboration, comment tracking, and autosave enhance productivity. The cloud-based nature means users can access and work on their documents from anywhere, which adds to efficiency.
- **M*emorability***: Returning users can easily remember how to navigate the software. The icons and menu structures are simple and consistent, ensuring users don't need to re-learn how to perform tasks.
- ***Error Frequency and Severity****:* Errors are minimal in Google Docs. Features like autosave prevent data loss in case of accidental closures or crashes. Additionally, the version history feature allows users to restore previous versions of documents, minimizing the impact of potential mistakes.
- ***Satisfaction****:* The minimalistic design and responsive interface provide a smooth user experience. The ability to collaborate in real-time, comment on changes, and access documents from multiple devices makes users feel satisfied with the overall interaction.

Key Usability Strengths:

- Simple, intuitive UI.
- Real-time collaboration is smooth and productive.
- Auto save and version history reduce user anxiety about data loss.

Usability Analysis of Microsoft Outlook

Overview: Microsoft Outlook is an email client and personal information manager that includes calendar, task management, contact management, and email features, widely used by businesses and professionals.

Usability Factors:

- **Learnability**: Outlook can be overwhelming for new users due to its many features and multi-pane layout. The UI is more complex than simpler email clients, so there is a steeper learning curve, especially for non-technical users. However, tutorials and helpful tooltips reduce the initial burden of learning.

- **Efficiency**: Once familiar with the interface, Outlook becomes a powerful tool for managing emails, scheduling meetings, and organizing tasks. Features like the calendar integration and email sorting rules help users streamline their workflows.
- **Memorability**: Outlook has a consistent layout with distinct sections for email, calendar, contacts, and tasks. Although its complexity may cause some memorability issues, users who interact with it frequently can remember the organization and features well.
- **Error Frequency and Severity**: While using Outlook, errors typically arise from misconfiguration of email accounts or difficulty in setting up features like rules or filters. These errors are not severe, and with online help or built-in troubleshooting guides, users can recover relatively easily.
- **Satisfaction**: Outlook's advanced features, such as integration with the Microsoft Office suite, extensive calendar features, and powerful search capabilities, provide significant satisfaction to power users. However, casual users might find it over-complicated and less user-friendly.

Key Usability Strengths:

- Excellent for business environments due to deep integrations with Microsoft Office tools.
- Advanced organizational tools, such as calendar and task management.
- Robust search feature to quickly locate emails and contacts.

References:

- Google Docs Official Documentation.: https://support.google.com/docs/
- Nielsen Norman Group. (2016). Real-Time Collaboration in Google Docs. https://www.nngroup.com/articles/real-time-collaboration/
- Microsoft Outlook Official Documentation. https://support.microsoft.com/en-us/outlook

ASSIGNMENT 3:

Problem Statement: List all feature of programming language and Write Programs to show how they help to write ROBUST code

Key Features of C Programming Language

1. ***Simplicity:*** C has a simple syntax with only 32 keywords. This simplicity helps developers write code that is easy to read and maintain.
2. ***Structured Language:*** C allows the creation of complex programs by breaking them down into simpler functions or modules. This supports modularity and reuse, enhancing maintainability and reliability.
3. ***Low-Level Memory Access:*** Through the use of pointers, C provides direct manipulation of memory addresses, making it ideal for system programming and performance-critical applications. This feature allows writing robust code with efficient memory usage.
4. ***Portability:*** C programs can run on different types of machines with minimal or no modifications, making it a highly portable language. Portability ensures that robust code works across platforms without being rewritten from scratch.
5. ***Rich Library Support:*** The C Standard Library provides a wide range of built-in functions, such as input/output, string manipulation, and math functions, which help in building robust applications.
6. ***Dynamic Memory Allocation:*** C supports dynamic memory allocation via functions like malloc(), calloc(), and free(). This feature enables writing robust and efficient programs by allocating memory at runtime based on the needs of the application.
7. ***Pointers:*** Pointers provide a powerful way to access and manipulate memory directly. They allow for efficient handling of arrays, strings, and dynamic data structures such as linked lists.
8. ***Recursion:*** C supports recursion, meaning that functions can call themselves. This feature is useful in situations requiring repetitive tasks or algorithms like divide-and-conquer.
9. ***Bitwise Manipulation:*** C provides operators for bitwise manipulation, making it easy to work with data at the binary level. This is critical for applications that require direct control over hardware or need to optimize memory usage.
10. ***Efficient Data Structures:*** C allows developers to create efficient data structures like arrays, linked lists, stacks, queues, trees, and graphs. These structures are essential for writing robust programs.

1. Structured Language and Modularity

```
#include <stdio.h>

// Function declaration
int factorial(int n);

int main() {
   int num;

   // Input
   printf("Enter a number: ");
   scanf("%d", &num);

   // Output
   if (num < 0)
      printf("Factorial of negative number doesn't exist.\n");
   else
      printf("Factorial of %d is %d\n", num, factorial(num));

   return 0;
}

// Function definition for factorial calculation
int factorial(int n) {
   if (n == 0)
      return 1;
   else
      return n * factorial(n - 1);
}
```

Sample input:

Enter a number: 5

Sample Output:

Factorial of 5 is 120

2. Support for Low-Level Memory Manipulation

```
#include <stdio.h>

// Function to swap two numbers using pointers
void swap(int *x, int *y) {
    int temp;
    temp = *x;  // dereferencing pointer x
    *x = *y;    // dereferencing pointer y
    *y = temp;
}

int main() {
    int a, b;

    // Input
    printf("Enter two numbers:\n");
    scanf("%d %d", &a, &b);

    printf("Before swapping: a = %d, b = %d\n", a, b);

    // Call swap function using pointers
    swap(&a, &b);

    printf("After swapping: a = %d, b = %d\n", a, b);

    return 0;
}
```

Sample Input:

Enter two numbers:
10 20

Sample Output:

Before swapping: a = 10, b = 20
After swapping: a = 20, b = 10

3. Memory Management and Dynamic Memory Allocation

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

int main() {
    int *arr;
    int n, i;

    // Input
    printf("Enter number of elements: ");
    scanf("%d", &n);

    // Dynamic memory allocation
    arr = (int*) malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory not allocated.\n");
        return 1;
    }

    // Taking input in dynamically allocated array
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Displaying the array elements
    printf("Array elements are:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    // Free allocated memory
    free(arr);

    return 0;
}
```

Sample Input:

Enter number of elements: 5
Enter 5 elements: 1 2 3 4 5

Sample Output:

Array elements are: 1 2 3 4 5

References:

1) C: The Complete Reference by Herbert Schildt:
2) **C Programming Tutorial:** https://www.cprogramming.com/tutorial/c-tutorial.html

ASSIGNMENT 4:

Problem Statement: Study the "assertions" in C language and its importance in writing reliable code study POSIX standard and write a C program under unix to show use of POSIX standard in writing portable code.

What are Assertions?

An **assertion** is a debugging tool used to check the assumptions made by the program during runtime. Assertions are statements that test conditions which should logically be true in a program. If the condition evaluates to false, the program terminates with an error message, which aids in identifying bugs early.

In C, assertions are implemented using the assert() macro from the <assert.h> header file. When the assertion fails, the program prints an error message and stops execution

Example Program Using Assertions:

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

int divide(int a, int b) {
    // Assert that the divisor should not be zero
    assert(b != 0);
    return a / b;
```

```
}
int main() {
   int x = 10, y = 2, z = 0;

   // This will work fine
   printf("Result of x/y: %d\n", divide(x, y));

   // This will trigger an assertion failure because z = 0
   printf("Result of x/z: %d\n", divide(x, z));
   return 0;}
```

Sample Output:

```
Result of x/y: 5
Assertion failed: b != 0, file assertions_example.c, line 6
Aborted
```

What is POSIX?

POSIX (Portable Operating System Interface) is a family of standards specified by the IEEE for maintaining compatibility between operating systems. POSIX defines the APIs, shell and utilities interfaces, and command-line interfaces for software compatibility with variants of Unix and other operating systems.

POSIX Importance:

- **Portability**: POSIX ensures that code can be ported across multiple Unix-like operating systems (Linux, macOS, FreeBSD, etc.) with minimal changes.
- **Standardized APIs**: It provides a standard interface for system calls like file operations, process management, and threading, ensuring consistency.
- **System Independence**: Programs developed under POSIX can run on any system that adheres to the POSIX standards.

POSIX and Portability in C Programs:

Programs written using POSIX-compliant APIs can be compiled and executed on any system supporting the POSIX standards, thereby ensuring portability across systems.

Example C Program Using POSIX API

The following example shows how to write a C program using POSIX APIs for file operations and process control on a Unix-based system.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int main() {
    int fd;
    char buffer[256];
    ssize_t bytesRead;

    // POSIX compliant file open: open the file in read-only mode
    fd = open("example.txt", O_RDONLY);

    if (fd == -1) {
        perror("Error opening the file");
        exit(EXIT_FAILURE);
    }

    // POSIX compliant read operation
    bytesRead = read(fd, buffer, sizeof(buffer) - 1);

    if (bytesRead == -1) {
        perror("Error reading the file");
        close(fd);
        exit(EXIT_FAILURE);
    }

    // Null-terminate the buffer for printing
```

```
    buffer[bytesRead] = '\0';

    // Print the content read from the file
    printf("File content: \n%s\n", buffer);

    // POSIX compliant file close
    if (close(fd) == -1) {
        perror("Error closing the file");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

*Steps to Compile and Run on a POSIX-compliant System (e.g., Unix/Linux)**:***

1. Save the program in a file, say posix_example.c.
2. Create a text file example.txt with some content for reading.
3. Compile the program using a POSIX-compliant compiler like gcc:

   bash

   ```
   gcc posix_example.c -o posix_example
   ```

4. Run the compiled program:

   ```
   ./posix_example
   ```

Sample Input (Content of example.txt):

Hello, POSIX standard!

Sample Output:

File content:
Hello, POSIX standard.