CHAPTER FOUR

SYSTEM IMPLEMENTATION AND TESTING

This chapter describes the system installation and requirements for *CodeIns* such as hardware, software and the operating system used. In addition, there are also other activities involved in system implementation such as coding, testing and documentation. The purpose of these activities is to convert the final physical system specifications into a working and reliable software and hardware and document the work that has been done.

4.1 Development Environment

Development environment has certain impact on the development of a system. Using suitable hardware and software would help to speed up system development. The hardware and software tools used to develop and document the entire system are discussed below.

4.1.1 Hardware Requirements

The requirements for server computer have been discussed in Section 3.4.1.

4.1.2 Software Requirements

The software requirements include those software tools for development, design and report writing.

4.1.2.1 Software Tools for Development

During the development of *CodeIns*, these software tools were used. Table 4.1 shows the software tools used to develop the system.

Table 4.1: Summary of Software Used

Software/Component	Description
Windows 95/98	Network operating system
Personal Web Server	Web-server service
Active Server Pages	Server Scripting Engine
Dynamic Hyper Text Markup Language	Coding the homepages
Microsoft Internet Explorer 5	Web browser

4.1.2.2 Software Tools for Design and Report Writing

There are a lot of software tools, which can be used in designing and writing report. The design process involves the drawing of structure charts, data flow diagrams and others that form the foundation of the software development. The purpose of this graphically logical design is to provide an overall view of the system and the interconnection between the modules. Microsoft Word was used during the design process and also for report writing.

Some screenshots of *CodeIns* are shown below.

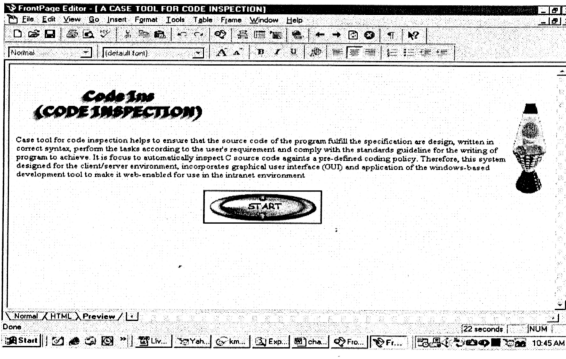

Figure 4.1: *CodeIns* Welcome Screen

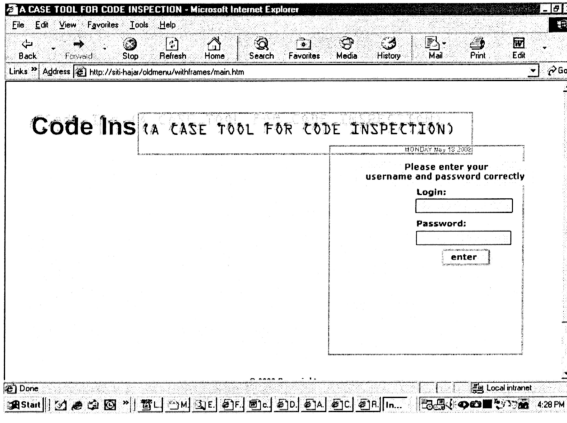

Figure 4.2: Login Page Screen

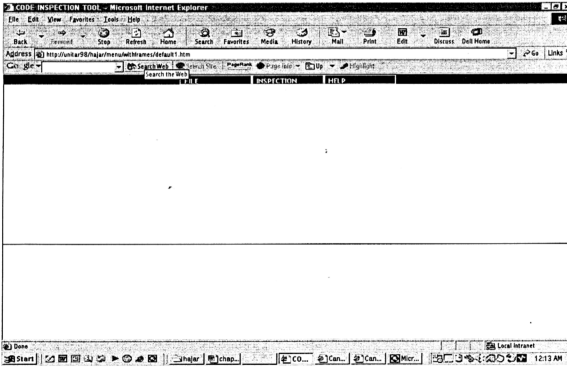

Figure 4.3: *CodeIns* Main Menu

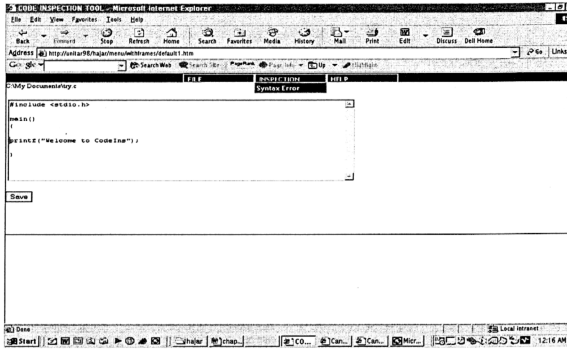

Figure 4.4: Inspect source code page

Other *CodeIns* screenshots are included in Appendix A. Sample coding are attached in Appendix B.

4.2 Testing

Testing is a process of exercising or evaluating a system by manual or automatic means to verify that it satisfies specification requirements or to identify the differences between the expected and actual results. A bug is an unexpected, questionable or undesired aspect or behaviour displayed, facilitated, or caused by the software being tested. Testing can uncover different classes of errors in a minimum amount of time and with a minimum amount of effort. The strategy used for this purpose is called system testing (Alan et al, 1998).

System testing tests all the implementation aspects of design. *CodeIns* was tested to determine whether it has met correctness attribute in terms of linkage to other module and able to show the list of syntax errors after the inspection process.

4.2.1 Testing Strategy

The proposed approach to testing the application was guided by several principles. By following these principles, a test process was developed. This process is generally applicable for testing an application with GUI interface. The test approach did not cover white box testing of the application codes in any depth. This approach only focused on the inner view of the application.

4.2.1.1 Test errors

The errors are categorized into types so that test cases can be designed to detect each type of the errors. Thus, making the testing more focus and eliminate duplication.

4.2.1.2 Test design techniques

The black box testing technique was used to test the C language source code.

4.2.2 Unit Testing

Unit testing was used to test each of the modules in *CodeIns*. The program module is tested in isolation to discover error in this method. All other modules are tightly integrated and therefore need to undergo integration testing. Table 4.2 show the valid User ID and password. Both User ID and password already set in the source code. Figure 4.5 shows the Login Screen and example is explained briefly in Table 4.3.

Table 4.2: Valid User ID and Password

User ID	**Password**
ct	hajar
ada	comel
mas	sri287

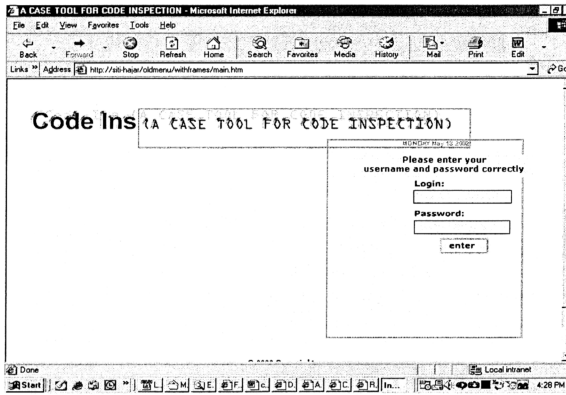

Figure 4.5: Login page screen

Table 4.3: Unit Testing Detail

Test	Expected Result	Actual
Valid User ID and password	No error message displayed. Proceed to Main Menu page (show in Figure 4.6)	As expected
Valid User ID and invalid password	Error message, prompt to login again (show in Figure 4.7)	As expected
Invalid User ID and valid password	Error message, prompt to login again (show in Figure 4.8)	As expected
Invalid User ID and invalid password	Error message, prompt to login again (show in Figure 4.9)	As expected

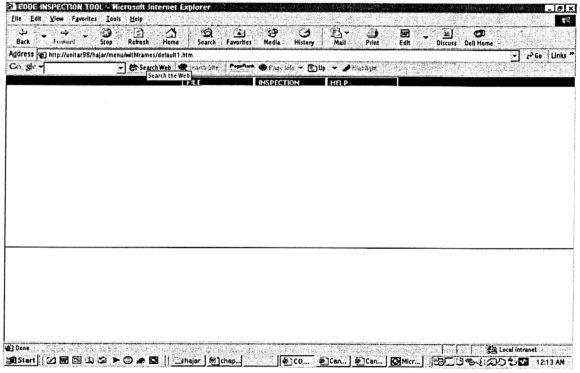

Figure 4.6: Unit test result – Valid User ID and password

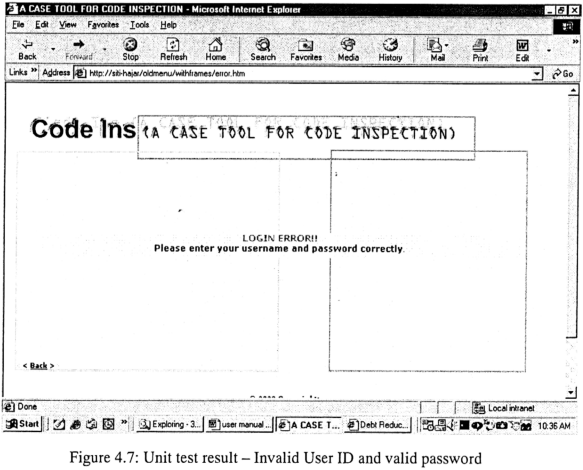

Figure 4.7: Unit test result – Invalid User ID and valid password

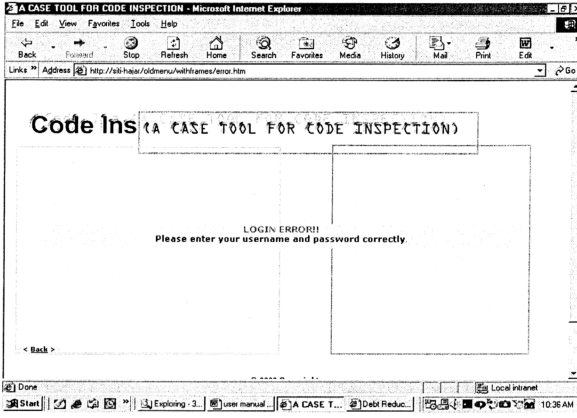

Figure 4.8: Unit test result – Valid User ID and invalid password

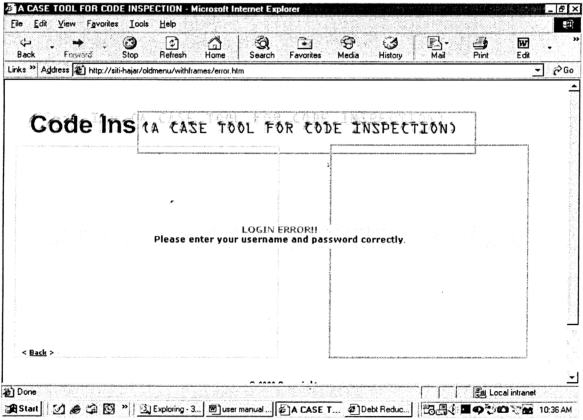

Figure 4.9: Unit test result – Invalid User ID and invalid password

4.2.3 System Testing

Having successfully tested the login screen unit testing, the next step is to test the system *CodeIns* module. Tests were performed to check that *CodeIns* could work properly under normal operating condition. These testing are divided into two different parts: program structure and syntax error. The expected results are shown in Table 4.4.

Table 4.4: System Testing Detail

Test	Expected Result	Actual
Program Structure		

1. Sequential
main()
{
printf("Welcome To ");
printf("CodeIns System);
}
	No errors displayed	As expected

2. Selection
main()
{
int a,b;
if (a > b)
printf(" %d is a maximum value", a);
else
printf(" %d is a minimum value", a);
}
	No errors displayed	As expected

3. Repetition
main()
{
int a=5,b=10;
while(a< b)
printf(" a = %d\n ", a);
a++;
}
	No errors displayed	As expected

Syntax Errors		
1. check typing error for keyword int Example: integer number;	Display syntax error undeclared identifier	As expected
2. check typing error for keyword float Example: float number;	Display syntax error undeclared identifier	As expected
3. check typing error for keyword char Example: character name;	Display syntax error undeclared identifier	As expected
4. check typing error for keyword while Example: whle (a>b)	Display syntax error undeclared identifier	As expected
5. check typing error for keyword if Example: ife (a>b)	Display syntax error undeclared identifier	As expected
6. check typing error for keyword for Example: For (a=0; a< b; a++)	Display syntax error undeclared identifier	As expected
7. check typing error for keyword case Example: switch (a) { case 1: break; cae 2: break; }	Display syntax error undeclared identifier	As expected

8. check typing error for keyword switch Example: Switch (a) { case 1: break; case 2: break; }	Display syntax error undeclared identifier	As expected
9. check typing error for keyword printf Example: Print("welcome to codeIns");	Display syntax error undeclared identifier	As expected
10. check typing error for keyword scanf Example: scan(" %d" , &num);	Display syntax error undeclared identifier	As expected

11. check typing error for keyword else
Example:
 	
if (a> b)
printf("a largest than b");
elsee
printf("a smallestthan b");
	Display syntax error undeclared identifier	As expected

12. check typing error for keyword return
Example:
 	
main()
{
printf("Welcome To ");
printf("CodeIns System ");
return0;
}
	Display syntax error undeclared identifier	As expected

13. check typing error for main
Example:
 	
main()
{
printf("Welcome To ");
printf("CodeIns System ");
return0;
}
	Display syntax error: invalid statement	As expected

14. check typing error for #include
Example:
 	
include <stdio.h>
main()
{
printf("Welcome To ");
printf("CodeIns System ");
return 0;
}
	Display syntax error: invalid statement	As expected

15. check for semicolon
Example:
 	
include <stdio.h>
main()
{
printf("Welcome To ")
printf("CodeIns System ");
return 0;
}
	Display syntax error: line printf("Welcome To ") missing semicolon	As expected

16. check for balance square bracket Example: int array[3 = { 1, 4, 5 };	Display syntax error: missing right square bracket	As expected
17. check for balance bracket Example: scan(" %d" , #	Display syntax error: missing right bracket	As expected

18. check for balance braces
Example:
 	
include <stdio.h>
main()
 	
printf("Welcome To ")
printf("CodeIns System");
return 0;
}
	Display syntax error: missing open braces	As expected

19. check for double quote
Example:
 	
include <stdio.h>
main()
 	
printf("Welcome To ")
printf(CodeIns System");
return 0;
}
	Display syntax error: Line printf(CodeIns System"); missing left of quote	As expected

Figure 4.10 shows the example of system testing, once the source codes to be inspected have been identified, the Open File module will open C the source codes in another frame. Inspection Module will then be chosen to detect the errors. If the source codes contain errors, a list of errors will appear at the third frame. The tester or user can edit the source codes.

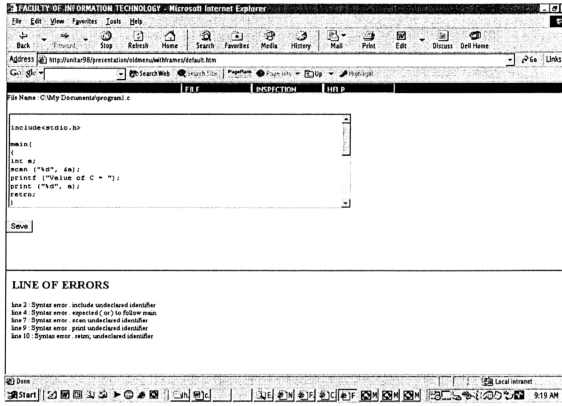

Figure 4.10: Example of system testing

4.2.4 Navigation Testing

Navigation testing can be categorized as a form of integration testing. Normally, new forms are created and tested in isolation. Integration of a new form into an application requires that the application menu definition and invocations of the window from other windows can be correctly implemented. This is done in the early stage of analysis and design. To conduct meaningful navigation testing, the following are created to be in place.

- An application backbone with at least the required menu options and call mechanisms to call the form under test
- Forms that can invoke the other form under test
- Forms that are called by other form under test

Table 4.5 shows the navigation on the forms in the proposed application.

Table 4.5: Forms available in *CodeIns*

Forms	**Able to go to**
Main menu	Access to all sub-modules
Open File	Open file module
Close File	Close the current process and link to blank frame
Exit File	Exit or close the window
Inspection	Inspection module
Help	Call the help contain module