Test Specification Document

DEVCLOUD Web Based Integrated Development Environment

TinTin

Alican Güçlükol

Anıl Paçacı

Meriç Taze

Serbay Arslanhan

Index

1. Introduction

CLOUD IDE is an application providing functionality of Integrated Development Environments over a distributed cloud resources. This project has two main components, namely server component and user interface components. Actually user interface is the main component, which will be the product delivered at the end of implementation and server is the component providing functionality to client at backend. To develop and durable, stable, well-integrated and reliable system, a complete test plan who will cover all the functionalities is required. The need of test plan is the purpose of this document. This TSR document is prepared for that purpose. In this document, after a brief introduction, we will present our test plan by describing overall testing strategy, and detailed test procedure including test tactics and test cases for the project. Then, information about testing resources and staffing, test work products, test record keeping, and test log will be given. Moreover, organization and responsibilities will be clearly explained. Finally, our test schedule for the whole test period will be described.

1.1 Goals and Objectives

As in the production process all of softwares, testing phase is important to ensure the quality and stability of the product delivered. To ensure these, we will test components of the software separately; later performance and integration test will be applied on final product.

1.2. Statement of Scope

This document is prepared for the specification of testing process of TinTin CLOUD IDE project. In this document, we mainly focus on:

- What is to be tested
- Constraints on test phase
- How bugs will be handled
- Testing Strategies and Procedures
- Responsible group members and testing schedule

1.3. Major Constraints

There are some main constraints which had effect on implementation of this product. These constraints are given below and they also have impact on the testing phrase.

Time:

Optimization of all modules for better performance and any possible speed up in the implementation has an essential importance because of the limited time for the project to be completed. So in DEVCLOUD - IDE, implementation and testing are done in parallel. In other words, whenever a new functionality is added to the system required tests are done immediately.

Data:

Since there are lots of different services passing data through each other and there are many different data needs to be saved on the server, any minimization on data is a data constraint goal.

Number of People:

Since there will be only one kind of users, which are the software developers, people with different characteristic involved in test phase will speed up implementation and improve quality of the final product.

Hardware:

On the client side, there will not any problem about the hardware requirements since nowadays almost any phone can render HTML5 pages. However, on the server side, there will be multiple users and requests at a time, capabilities of server hardware is of importance and should not be unnoticed during test phase.

1.4. Definitions, Acronyms and Abbreviations

IDE: Integrated Development Environment

CLI: Command Line Interface

TSR: Test Specifications Report

TRAC: Trac is an open source, web-based project management and bug-tracking tool.

1.5. References

Test Specification Template, METU Computer Engineering, Spring 2012

Pressman, Roger S. Software Engineering: A Practitioner's Approach, Sixth edition. New York, NY: McGraw-Hill

IEEE Standard for Software Test Documentation

Detailed Design Report prepared by TinTin

2. Updates

Section Name	Section Number	Status
Workspace Explorer Functional Test Cases	4.2	new test added
Code Editor Functional Test Cases	4.3	new test added
CLI Functional Test Cases	4.4	new test added
Build-Run Functional Test Cases	4.5	new test added

3. Test Plan

The purposes of these tests are to test the product whether it accomplishes the requirements that are implemented. The main objective is to test the functionality of the product. Also the detailed objective for each test case is described. In this process certain inputs are applied to software and responses to these inputs as outputs are tested. In these tests both positive and negative scenario tests are considered and run on software. The test cases are categorized as below:

Test Number	Test Name
1	Authentication Functional Test Cases
2	Workspace Explorer Functional Test Cases
3	Code Editor Functional Test Cases
4	CLI Functional Test Cases
5	Build – Run Functional Test Cases

4. Test Cases

In this section, test cases which are grouped in previous section will be described in detail.

4.1 Authentication Functional Test Cases

Test Case ID	Authentication – 01
Test Case Name	Authentication – User registration with Invalid Parameters
Test Type	Functional Test
Test Case Description	Trying to register to system with missing parameters
Test Case Objective	To test whether software registration process is capable of applying conditions for user registration
Input	User Input
Pre-Conditions	Browser directed to Login page
Steps	1. Open home page 2. Click Register button 3. Enter all parameters, e-mail 4. Click Register button
Expected Response	An alert stating that required field(s) missing
Post Conditions	Stay in login Page

Test Case ID	Authentication – 02
Test Case Name	Authentication – User registration with Valid Parameters
Test Type	Functional Test
Test Case Description	Trying to register to system with valid parameters
Test Case Objective	To test whether software registration process is capable of applying conditions for user registration
Input	User Input
Pre-Conditions	Browser directed to Login page
Steps	1. Open home page 2. Click Register button 3. Enter all parameters 4. Click Register button
Expected Response	An alert stating that registration is complete
Post Conditions	User directed to main screen of product

Test Case ID	Authentication – 03
Test Case Name	Authentication – User Sign in with Invalid Credentials
Test Type	Functional Test
Test Case Description	Trying to login to system with invalid credentials
Test Case Objective	To test whether software registration process is capable of applying conditions for user authentication
Input	User credentials
Pre-Conditions	Browser directed to Login page
Steps	1. Open home page 2. Enter all parameters as random strings

	4. Click Login button
Expected Response	An alert stating that invalid credentials
Post Conditions	Stay in login Page

Test Case ID	Authentication – 04
Test Case Name	Authentication – User registration with Valid Parameters
Test Type	Functional Test
Test Case Description	Login system with correct credentials
Test Case Objective	To test whether software registration process is capable of applying conditions for user registration
Input	User credentials
Pre-Conditions	Browser directed to Login page, User already registered
Steps	1. Open home page 2. Enter all parameters credentials 3. Click Login button
Expected Response	Re-direction to product main page
Post Conditions	User logged in system
Test Case ID	Authentication – 05
Test Case Name	Authentication – User Logout
Test Type	Functional Test
Test Case Description	Trying to logout from system
Test Case Objective	To test whether software allows to logout of the system
Input	User Input
Pre-Conditions	User successfully logged in
Steps	1. Open home page 2. Login system 3. Stay in main page 4. Click Logout button
Expected Response	An alert stating that user successfully logged out
Post Conditions	Re-direction to Login Page

4.2 Workspace Explorer Functional Test Cases

Test Case ID	Workspace Explorer – 01
Test Case Name	Workspace Explorer – Create file
Test Type	Functional Test
Test Case Description	Trying to create a new file
Test Case Objective	To test whether software allows to create a new file
Input	User Input
Pre-Conditions	User successfully logged in
Steps	1. Open main page 2. Click Menu 3. Click New File 4. Write test.txt to dialog 5. Click OK
Expected Response	A file with name test.txt must be created and opened
Post Conditions	Give focus to created file

Test Case ID	Workspace Explorer – 02
Test Case Name	Workspace Explorer – Open file
Test Type	Functional Test
Test Case Description	Trying to open a file
Test Case Objective	To test whether software allows to open a file
Input	User Input
Pre-Conditions	User successfully logged in At least one file must be exist
Steps	1. Open main page 2. Click a file from the file explorer
Expected Response	Selected file must be opened in a new tab
Post Conditions	Give focus to opened file

Test Case ID	Workspace Explorer – 03
Test Case Name	Workspace Explorer – Save file
Test Type	Functional Test
Test Case Description	Trying to save a file
Test Case Objective	To test whether software allows to save a file
Input	User Input
Pre-Conditions	User successfully logged in At least one file must be exist
Steps	1. Open main page 2. Click a file from the file explorer 3. Add some text 4. Click to File > Save File 5. Close and reopen the file
Expected Response	Reopened file content must be the same with closed one
Post Conditions	-

Test Case ID	Workspace Explorer – 04
Test Case Name	Workspace Explorer – Close file
Test Type	Functional Test
Test Case Description	Trying to close a file
Test Case Objective	To test whether software allows to close a file
Input	User Input
Pre-Conditions	User successfully logged in At least one file must be exist
Steps	1. Open main page 2. Click a file from the file explorer 3. Click to File >Close
Expected Response	File tab must be closed.
Post Conditions	Give focus to previous tab

Test Case ID	Workspace Explorer – 05
Test Case Name	Workspace Explorer – New file
Test Type	Functional Test
Test Case Description	Trying to create new file
Test Case Objective	To test whether software allows to create a new file
Input	User Input
Pre-Conditions	User successfully logged in
Steps	1. Open main page 2. Right click a folder from the file explorer 3. Click to New File 4. Write file name 5. Click ok
Expected Response	New file must be created with the given name
Post Conditions	Refresh file tree to show new file

Test Case ID	Workspace Explorer – 06
Test Case Name	Workspace Explorer – New folder
Test Type	Functional Test
Test Case Description	Trying to create new folder
Test Case Objective	To test whether software allows to create a new folder
Input	User Input
Pre-Conditions	User successfully logged in
Steps	1. Open main page 2. Right click a folder from the file explorer 3. Click to New Folder 4. Write folder name 5. Click ok
Expected Response	New folder must be created with the given name
Post Conditions	Refresh file tree to show new folder

Test Case ID	Workspace Explorer – 07
Test Case Name	Workspace Explorer – Build
Test Type	Functional Test
Test Case Description	Trying to build a project
Test Case Objective	To test whether software allows build a project
Input	User Input
Pre-Conditions	User successfully logged in
Steps	1. Open main page 2. Right click a folder which contains a Makefile from the file explorer 3. Click to Build
Expected Response	Folder must be built according to Makefile
Post Conditions	Refresh file tree to show new folder

4.3 Code Editor Functional Test Cases

Test Case ID	Code Editor – 01
Test Case Name	Code Editor – Syntax Highlighting
Test Type	Functional Test
Test Case Description	Code editor syntax highlighting feature for the specified programming language
Test Case Objective	To test whether code editor is capable of highlighting the keywords, variables, data types correctly
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Create a sample C source file 3. Open the file with the code editor
Expected Response	The code will be highlighted according to the C programming language syntax
Post Conditions	

Test Case ID	Code Editor – 02
Test Case Name	Code Editor – Auto Indentation
Test Type	Functional Test
Test Case Description	Code editor auto indentation feature for the specified programming language
Test Case Objective	To test whether code editor is capable of indenting the code with respect to the current position of the cursor
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Create a sample C source file 3. Open the file with the code editor
Expected Response	The code will be indented automatically when the user presses 'Enter' key after writing a line of code.
Post Conditions	

Test Case ID	Code Editor – 03
Test Case Name	Code Editor – Bracket/Brace matching
Test Type	Functional Test
Test Case Description	Code editor bracket/brace matching feature
Test Case Objective	To test whether code editor is capable of matching the brackets and braces with their pairs
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Create a sample C source file 3. Open the file with the code editor

	4. Place the cursor on a bracket or brace
Expected Response	The matching bracket or brace will be shown within a square encloses it.
Post Conditions	

Test Case ID	Code Editor – 04
Test Case Name	Code Editor – Auto-completion
Test Type	Functional Test
Test Case Description	Code editor auto-completion feature
Test Case Objective	To test whether code editor is capable of providing auto-completion options for a given word
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Create a sample C source file 3. Open the file with the code editor 4. Type some part of the word that will be completed 5. Press Ctrl+Space keys
Expected Response	A set of options that are available in the file for the given word will be shown in a box under the cursor.
Post Conditions	

Test Case ID	Code Editor – 05
Test Case Name	Code Editor – Find
Test Type	Functional Test
Test Case Description	Code editor find feature
Test Case Objective	To test whether code editor is capable of finding a given word in the source file
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Create a sample C source file 3. Open the file with the code editor 4. Click the Find link in the Edit menu 5. Type the word that will be searched in the document into the provided input field
Expected Response	The matching occurrences of the given word will be enclosed within a square if there is any.
Post Conditions	

Test Case ID	Code Editor – 06
Test Case Name	Code Editor – Replace
Test Type	Functional Test
Test Case Description	Code editor replace feature

Test Case Objective	To test whether code editor is capable of replacing a given word with the given replacement word in the source file
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Create a sample C source file 3. Open the file with the code editor 4. Click the Replace link in the Edit menu 5. Type the word that will be searched in the document into the provided input field 6. Type the word that will be replaced with the found occurrences of the searched word in the document into the provided input field
Expected Response	The matching occurrences of the given word will be replaced with the given word.
Post Conditions	

Test Case ID	Code Editor – 07
Test Case Name	Code Editor – Editor Themes
Test Type	Functional Test
Test Case Description	Code editor multiple editor themes feature
Test Case Objective	To test whether code editor is capable of providing multiple editor themes
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Click the Options link in the Edit menu 3. Choose a different theme from the combo-box provided 4. Open a file with the code editor
Expected Response	The selected editor theme will be applied to the editors opened after this point.
Post Conditions	

Test Case ID	Code Editor – 08
Test Case Name	Code Editor – Displaying line numbers
Test Type	Functional Test
Test Case Description	Code editor displaying line numbers feature
Test Case Objective	To test whether code editor is capable of displaying the corresponding line numbers in the source file
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Login to the system 2. Create a sample C source file 3. Open the file with the code editor
Expected Response	The corresponding line numbers for each line in the source file will be

	displayed in the gutter part of the editor.
Post Conditions	

Test Case ID	Code Editor – 09
Test Case Name	Code Editor – Type Navigator
Test Type	Functional Test
Test Case Description	Code editor generates type navigator for opened file on the editor and show this navigator on the left side of the editor
Test Case Objective	To test whether code editor is capable of generating type navigator and its functionality correctly
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file and opened it with the code editor.
Steps	1. Click “Edit->Show type navigator” from top menu 2. Click on a function from the navigator
Expected Response	The navigator will be showing all functions and the cursor will go to the line where function defined and that line will be highlighted
Post Conditions	

Test Case ID	Code Editor – 10
Test Case Name	Code Editor – Navigate to function
Test Type	Functional Test
Test Case Description	When Alt+F3 typed while the cursor is on a function name cursor moves to the line where that function is defined. If the function is defined on another file first that file is opened
Test Case Objective	To test whether code editor is capable to process Alt+F3 short-cut correctly
Input	Sample C code written in the editor
Pre-Conditions	User have logged in, created a C source file, opened it with the code editor and cursor is on a function name
Steps	1. Type Alt+F3
Expected Response	If the definition is on another file first that file will be opened. The cursor goes to the line where the function is defined
Post Conditions	

4.4 CLI Functional Test Cases

Test Case ID	CLI – 01
Test Case Name	CLI – Running a valid command
Test Type	Functional Test
Test Case Description	Trying to run a valid Unix command with valid parameters on terminal
Test Case Objective	To test whether the software is able to run Unix commands and give appropriate response
Input	User Input
Pre-Conditions	1. Browser directed to the software page 2. CLI is opened on the page
Steps	1. Type a valid command on the command line 2. Press enter
Expected Response	Result of the command shown in terminal or user's workspace depending the command
Post Conditions	Command line is ready for a new command

Test Case ID	CLI – 02
Test Case Name	CLI – Running an invalid command
Test Type	Functional Test
Test Case Description	Trying to run an invalid Unix command on terminal
Test Case Objective	To see the response of system to an invalid command
Input	User Input
Pre-Conditions	1. Browser directed to the software page 2. CLI is opened on the page
Steps	1. Type an invalid command on the command line 2. Press enter
Expected Response	An error message is shown in command line which indicates that there is no such a command
Post Conditions	Command line is ready for a new command

Test Case ID	CLI – 03
Test Case Name	CLI – Running a valid command with invalid parameters
Test Type	Functional Test
Test Case Description	Trying to run a valid command with invalid parameters on terminal
Test Case Objective	To see the response of system to invalid parameters
Input	User Input
Pre-Conditions	1. Browser directed to the software page 2. CLI is opened on the page
Steps	1. Type a valid command but invalid parameters on the command line 2. Press enter
Expected Response	An error message is shown in command line which indicates that parameters are invalid and gives information about valid parameters
Post Conditions	Command line is ready for a new command

Test Case ID	CLI – 04
Test Case Name	CLI – Going back on the command history
Test Type	Functional Test
Test Case Description	Trying to view past commands using keyboard
Test Case Objective	To test whether past commands could be viewed by pressing "up" key
Input	Keyboard event
Pre-Conditions	1. User successfully logged in 2. CLI is opened on the page 3. At least one command is executed after CLI is opened 4. Cursor is on the command line
Steps	1. Press "up" key from keyboard
Expected Response	Command executed just before the current command is shown on the command line
Post Conditions	

Test Case ID	CLI – 05
Test Case Name	CLI – Going forward on the command history
Test Type	Functional Test
Test Case Description	Trying to view commands newer than the command which is being viewed currently
Test Case Objective	To test whether newer commands could be viewed by pressing "down" key
Input	Keyboard event
Pre-Conditions	1. User successfully logged in 2. CLI is opened on the page 3. At least one command is executed after CLI is opened 4. A past command is viewed by using "up" key
Steps	1. Press "down" key from keyboard
Expected Response	Command executed just after the current command is shown on the command line
Post Conditions	

Test Case ID	CLI – 06
Test Case Name	CLI – Viewing available commands
Test Type	Functional Test
Test Case Description	Trying to view available commands which includes the phrase typed so far
Test Case Objective	To test whether available matching commands could be listed on terminal by pressing "tab" key on keyboard
Input	Keyboard event
Pre-Conditions	1. User successfully logged in 2. CLI is opened on the page 3. Cursor is on the command line
Steps	1. Press "tab" key from keyboard
Expected Response	Available matching command are listed on terminal
Post Conditions	Command line returns the situation before "tab" key is pressed

Test Case ID	CLI – 07
Test Case Name	CLI – Point Home Directory
Test Type	Functional Test
Test Case Description	CLI starts at home directory by default
Test Case Objective	To ensure that whenever new CLI is opened, it points to the users home directory
Input	Click event on terminal
Pre-Conditions	1. User successfully logged in 2. CLI is opened on the page
Steps	1. Open Terminal Window
Expected Response	Terminal will point to users home directory
Post Conditions	Ready terminal at users home directory

Test Case ID	CLI – 07
Test Case Name	CLI – Permissions on System Folder
Test Type	Functional Test
Test Case Description	Trying to modify system files
Test Case Objective	To ensure user is privileged to read and execute system files, but not modify them
Input	Keyboard event
Pre-Conditions	1. User successfully logged in 2. CLI is opened on the page 3. Cursor is on the command line 4. cd into root directory 5. try to remove existing directory
Steps	1. type “cd /” 2. type “rm –rf usr”
Expected Response	User can change directory but not remove file
Post Conditions	Command line returns “permission denied” error

Test Case ID	CLI – 08
Test Case Name	CLI – Permissions on other users files
Test Type	Functional Test
Test Case Description	Trying to modify other users files
Test Case Objective	To ensure that user does not have any permissions on other users files
Input	Keyboard event
Pre-Conditions	1. User successfully logged in 2. CLI is opened on the page 3. Cursor is on the command line 4. cd into upper directory 5. try to navigate into other users directory
Steps	1. Type “cd ..” 2. Type “ls” and see other users folders 3. Type “cd <folder name>”
Expected Response	User will not be able to navigate other users directory
Post Conditions	Command line returns the “can’t cd into <folder name>” error

4.5 Build- Run Functional Test Cases

Test Case ID	Build-Run – 01
Test Case Name	Build-Run – Build with unsupported file type
Test Type	Functional Test
Test Case Description	Trying to build a unsupported file type
Test Case Objective	To test whether software allows to build a unsupported file type
Input	User Input
Pre-Conditions	User successfully logged in An unsupported file must exist in workspace
Steps	1. Open main page 2. Select the file from workspace explorer 3. Click Build button
Expected Response	An alert stating that build is not supported for that file type
Post Conditions	Give focus to editor

Test Case ID	Build-Run – 02
Test Case Name	Build-Run – Build when Terminal tab focused
Test Type	Functional Test
Test Case Description	Trying to build when a terminal tab is focused
Test Case Objective	To test whether software handle the build on a terminal tab
Input	User Input
Pre-Conditions	User successfully logged in
Steps	1. Open main page 2. Open a new terminal from the menu 3. Click Build button
Expected Response	An alert stating that build cannot be done on a terminal tab
Post Conditions	Give focus to terminal

Test Case ID	Build-Run – 03
Test Case Name	Build-Run – Build without file
Test Type	Functional Test
Test Case Description	Trying to build when there is no opened file
Test Case Objective	To test whether software allows to build when there is no opened file
Input	User Input
Pre-Conditions	User successfully logged in
Steps	1. Open main page 2. Close all tabs if exists 3. Click Build button
Expected Response	An alert stating that build cannot be done if there is no opened file
Post Conditions	Give focus to page

Test Case ID	Build-Run – 04
Test Case Name	Build-Run – Build with supported file type
Test Type	Functional Test
Test Case Description	Trying to build a supported file type
Test Case Objective	To test whether software allows to build a supported file type
Input	User Input
Pre-Conditions	User successfully logged in A supported file must exist in workspace
Steps	1. Open main page 2. Select the file from workspace explorer 3. Click Build button
Expected Response	An alert stating that build is completed successfully
Post Conditions	Open a terminal tab, and show the output

Test Case ID	Build-Run – 05
Test Case Name	Build-Run – Run an executable built before
Test Type	Functional Test
Test Case Description	Run an executable built before
Test Case Objective	To test whether software allows to run executable properly
Input	User Input
Pre-Conditions	User successfully logged in An executable must exist in workspace
Steps	1. Open main page 2. Select the executable from workspace explorer to run 3. Click Run button
Expected Response	A new terminal appears as a tab that is used to allow the user to enter input and see the output
Post Conditions	The program exits

Test Case ID	Build-Run – 06
Test Case Name	Build-Run – Run an executable that tries to open a file located outside the workspace
Test Type	Functional Test
Test Case Description	Run an executable that tries to open a file located outside of the user workspace
Test Case Objective	To test whether application limits the user’s access to other users’ workspaces
Input	-
Pre-Conditions	User successfully logged in An executable must exist in workspace
Steps	1. Open main page 2. Select the executable that tries to open a file located outside of the user’s workspace from workspace explorer to run 3. Click Run button
Expected Response	The program aborts because it does not have the permission to open the files that are located outside of the user’s workspace
Post Conditions	The program exits

Test Case ID	Build-Run – 07
Test Case Name	Build-Run – Set breakpoints in debug mode
Test Type	Functional Test
Test Case Description	Set breakpoints for an executable
Test Case Objective	To test whether the user is able to stop the program execution by setting breakpoints
Input	-
Pre-Conditions	User successfully logged in An executable must exist in workspace
Steps	1. Open main page 2. Set breakpoints by clicking on the gutter part of the editor 3. Select the executable from workspace explorer to debug 4. Click Debug button
Expected Response	The program stops execution and waits for the user's command to continue at the given breakpoint
Post Conditions	The user continues debugging

Test Case ID	Build-Run – 08
Test Case Name	Build-Run – Continue in debug mode
Test Type	Functional Test
Test Case Description	Continue execution for an executable that is stopped at the given breakpoint
Test Case Objective	To test whether the user is able to make the program continue from a breakpoint that is set before
Input	-
Pre-Conditions	User successfully logged in An executable must exist in workspace
Steps	1. Open main page 2. Set breakpoints by clicking on the gutter part of the editor 3. Select the executable from workspace explorer to debug 4. Click Debug button 5. When the program stops, click Continue button.
Expected Response	The program continues execution
Post Conditions	The user continues debugging

Test Case ID	Build-Run – 09
Test Case Name	Build-Run – Step in debug mode
Test Type	Functional Test
Test Case Description	Continue one more step to execution for an executable that is stopped at the given breakpoint
Test Case Objective	To test whether the user is able to make the program step a line from a breakpoint that is set before
Input	-
Pre-Conditions	User successfully logged in An executable must exist in workspace
Steps	1. Open main page 2. Set breakpoints by clicking on the gutter part of the editor

	3. Select the executable from workspace explorer to debug 4. Click Debug button 5. When the program stops, click Step button.
Expected Response	The program steps one line and waits for further commands
Post Conditions	The user continues debugging

5. Test Resources and Staffing

All the functional tests described above will be tested by all group members to have a common opinion about the product functionalities. Also testes will be applied by non-developers as well to have an unbiased opinion about the product. However, preparation and evaluation of the tests will be done as follow:

Test Name	**Responsible Member**
Authentication Tests	Anıl Paçacı
Workspace Tests	Meriç Taze
Code Editor Tests	Serbay Arslanhan – Alican Güçlükol
CLI Tests	Alican Güçlükol
Build – Run Tests	Meriç Taze – Serbay Arslanhan

6. Test Schedule

Below table shows a detailed scheduling of the tests.

Test Name	**Deadline**	**Final Demo Deadline**
Authentication Tests	30.04.2013	30.04.2013
Workspace Tests	05.05.2013	30.05.2013
Code Editor Tests	15.05.2013	02.06.2013
CLI Tests	15.05.2013	06.06.2013
Build – Run Tests	05.05.2013	06.06.2013