COVER PAGE

Tilted Programmers

Kevin Le

Michael Ha

David Luong

David Tran

Jonathan Peng

Stephen Chan

4/27/16

Iteration 3

Introduction to Software Engineering

Professor Ehteshami

**2. Revision History**

**Iteration 3**

**April 10th**

Michael= Present

David L= Present

David T= Not Present

Stephen= Present

Kevin= Present

Jonathan= Not Present

Group created a To do document in order to discuss what needs to be done. Only 1 use case will be happening which is the saving the user’s chapter scores. Group members were assigned specific tasks for the iteration.

**April 16th**

Michael= Present

David L= Present

David T= Not Present

Stephen= Present

Kevin= Present

Jonathan= Present

Group update on mockup and timeline for iteration 3. Delegated more assignments and worked on the Unit Test frameworks as well as automation testing. Finalized tests for the iteration.

Table of Contents

Content: Page Number:

Revision History...................................................................................................................1

Table of Contents…………………………………………………………………………..2

Pre-Game Planning...............................................................................................................3

Use Case Diagram.................................................................................................................4

Use Cases..............................................................................................................................5

Staging Grooming.................................................................................................................5

User Story..............................................................................................................................5

Sprint Backlog Screenshots...................................................................................................6-8

Explanation: System & Documented Activities....................................................................9

Class Diagram........................................................................................................................9

System Sequence Diagram………………………………………………………………….10

Communication Diagram.......................................................................................................10

Test Plan, Test Suite, Test Cases............................................................................................13-17

Automation…..……………………………………………………………………………...18-27

Code for functional user stories………………………………………………………..…...28-29

User Manual............................................................................................................................29-32

References...............................................................................................................................33

Team Charter...........................................................................................................................33-35

Team Evaluation……………………………………………………………………………..36

Software Engineering: A Practitioner’s Approach:

Chapter 3: Software Process Structure..................................................................................30-40

Chapter 4: Process Models...................................................................................................40-66

Chapter 5: Agile Development...............................................................................................66-87

Chapter 8: Understanding Requirements............................................................................131-166

Chapter 9: Requirements Modeling: Scenario-Based Methods..........................................166-184

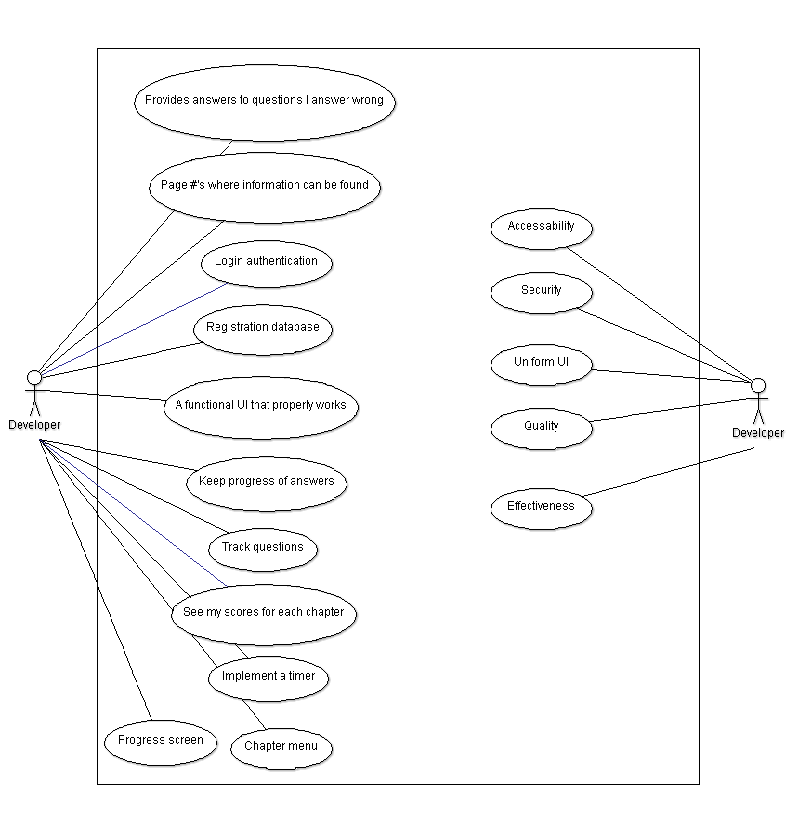
Chapter 10: Requirements Modeling: Class- Based Methods.............................................184-202

**Explain how your team worked on Pre-game Planning**

**Pre Game Planning**

For this Iteration, our primary focus was on creating a database of scores for the user, improving on previous features and providing automation for future testing. By creating a database of scores for the user, they will be able to see the improvement they have made since they have started using our tool. By adding a new beeping feature for our timer, users can know how much time they have before the test is over. We also added colors to the ui to make it more aesthetically pleasing to everyone. We were able to measure how much time these actions would take with the use of planning poker. Afterwards, mockups were created so our developers could start working on these user stories.

**3. Use Case Diagram**



**4. Use-case (textual)**

## **Use Case:** Viewing Chapter Scores

**ID :** UC\_13

**Description:** The user should be able to view their chapter scores.

**Level: High**

**Primary Actor:** Students, professors

**Supporting Actors:**

**Precondition:** User has already taken the test for one of the chapters

## **Main Success Scenario**

1. User goes to chapter select menu.
2. User clicks on the chapter scores button.
3. User sees all of their scores for each chapter with the date.

## **Extensions**

## **Use Case:** Writing Chapter Scores to file

**ID :** UC\_14

**Description:** The program writes a chapter score to the text file to be saved and read later.

**Level:** High

**Primary Actor:** Students, professors

**Supporting Actors:**

**Precondition:** User has already taken the test for one of the chapters

## **Main Success Scenario**

1. User goes to chapter select menu.
2. User takes the choices for the quiz.
3. User sees calculated score.
4. Score is written to file.

## **Extensions**

**Staging - Grooming** In our Iteration 3 of Staging, we wanted to address the minor improvements the stakeholder presented to us regarding the previous iterations. We focused on upgrading our UI by increasing the font and implementing colors based upon our stakeholders recommendations. We also wanted to save the user’s scores to their account and designed a way to save them while still using our text file with the database of accounts. We added timer improvements such as notification near the end of the limit alerting the user to finish the quiz. As for prioritizing each task, we decided to focus heavily on saving the user’s score since it was a high priority. Each of the minor UI improvements were considered low priorities and implemented with fe issues.

**5. User Story**

**US-12: Save User’s chapter scores**

As a user, I would like my chapter score saved in order to compare it to my other quizzes from the other chapters.

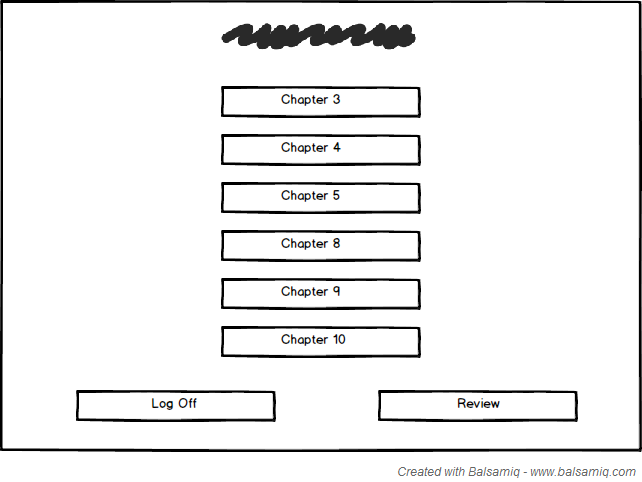
Relation to Use Case: Relates to chapter select screen as well as results screen.

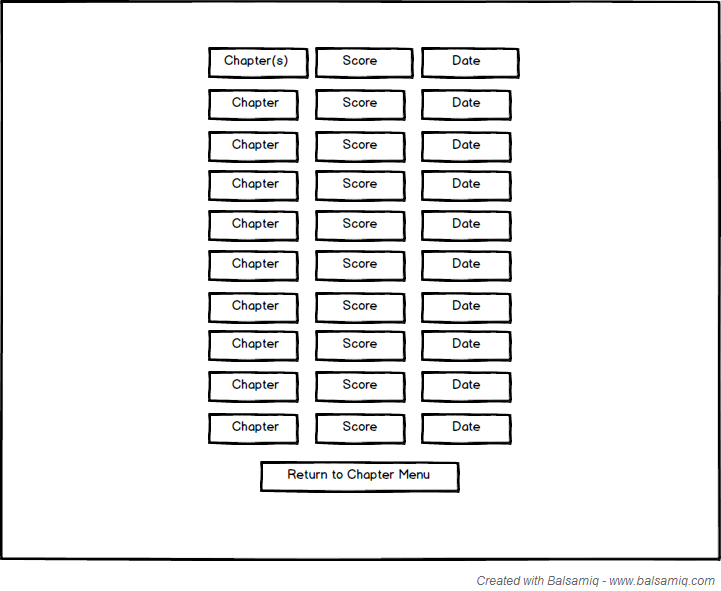
Assigned Developer: Kevin Le, David Tran, Jonathan Peng

Due Date: 4/25/16

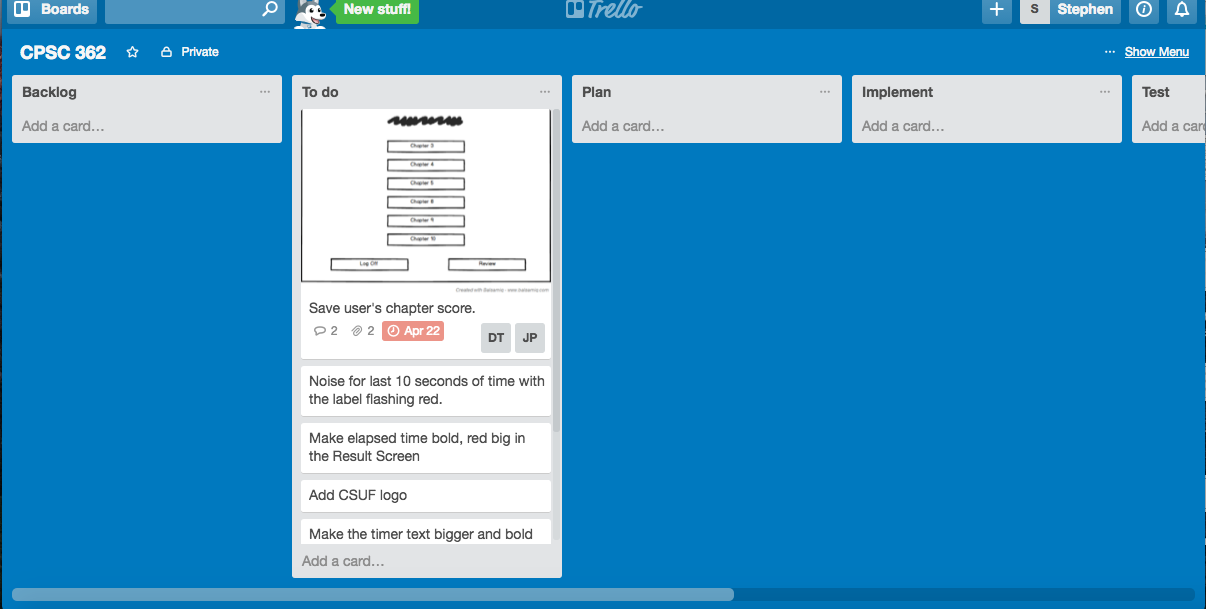
Priority: High

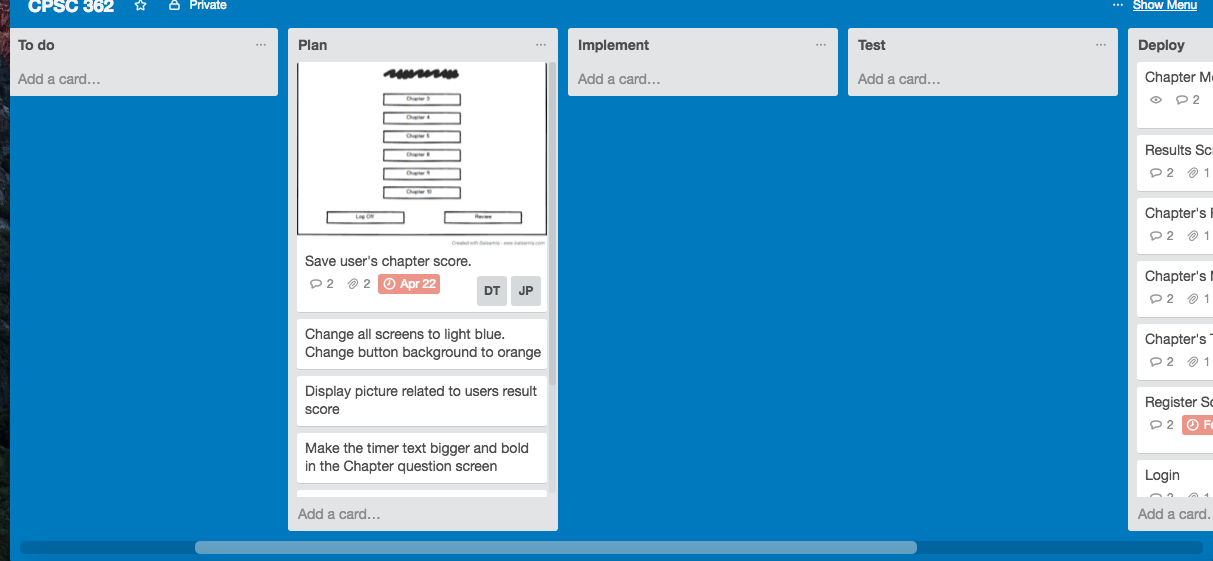
MockUp:

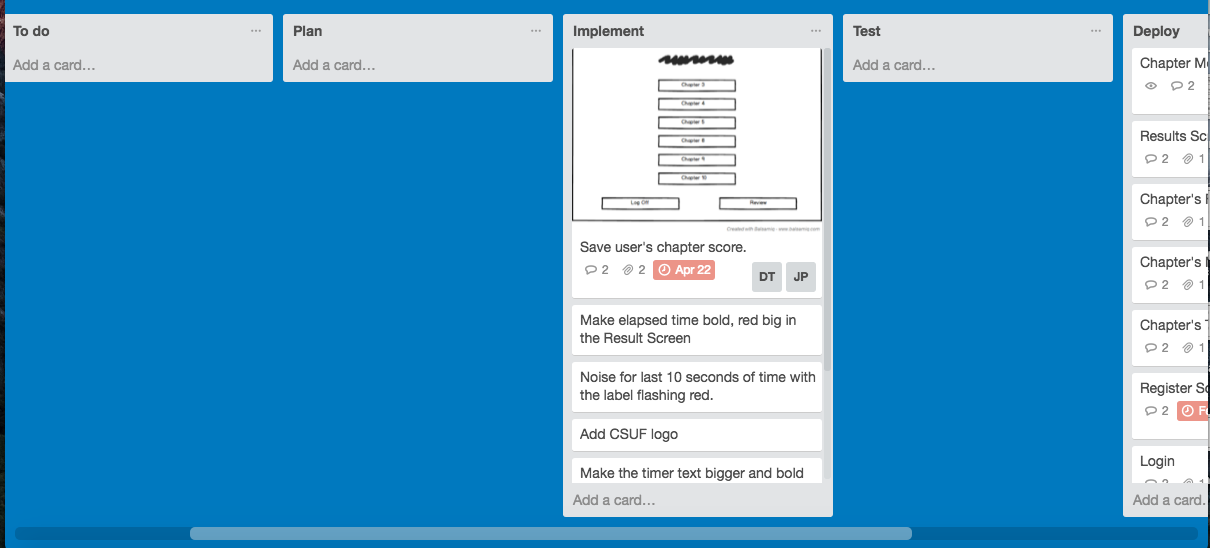


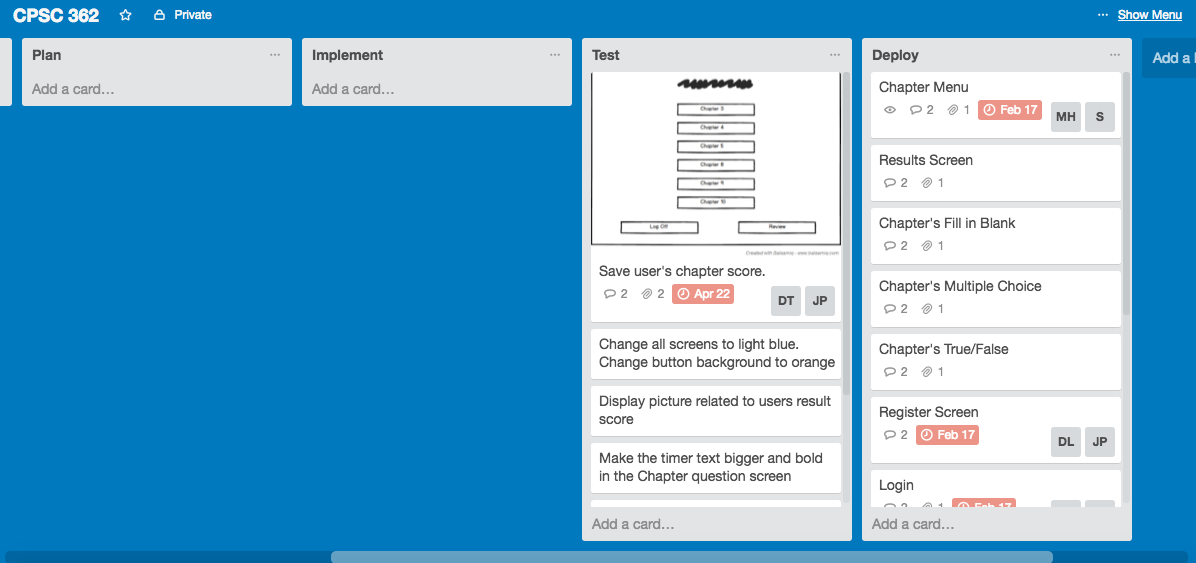


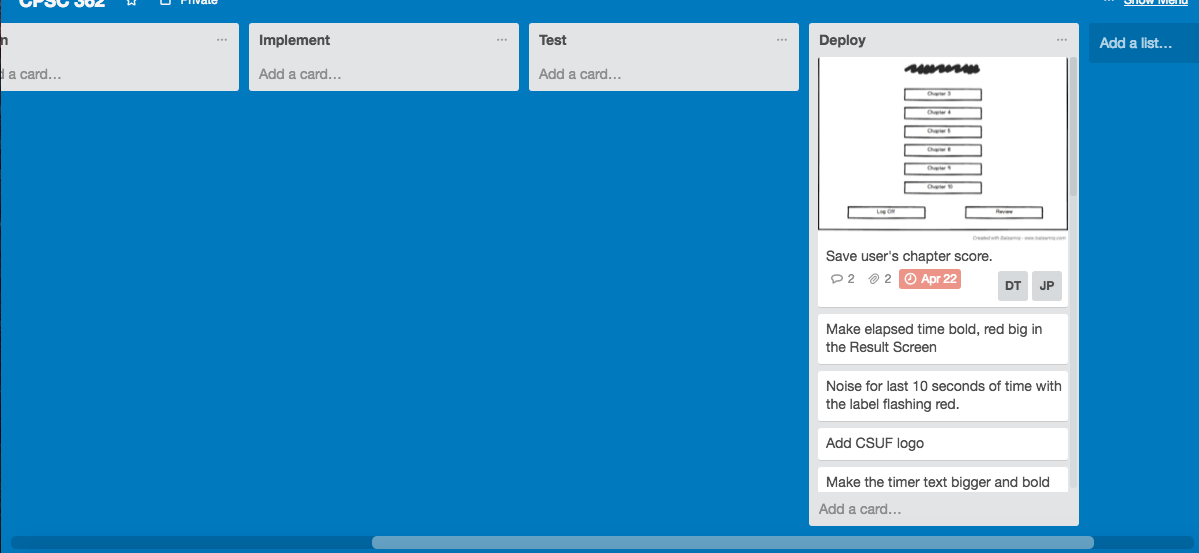
**6. Sprint Backlog Screenshots of your Trello Kanban-Board.**







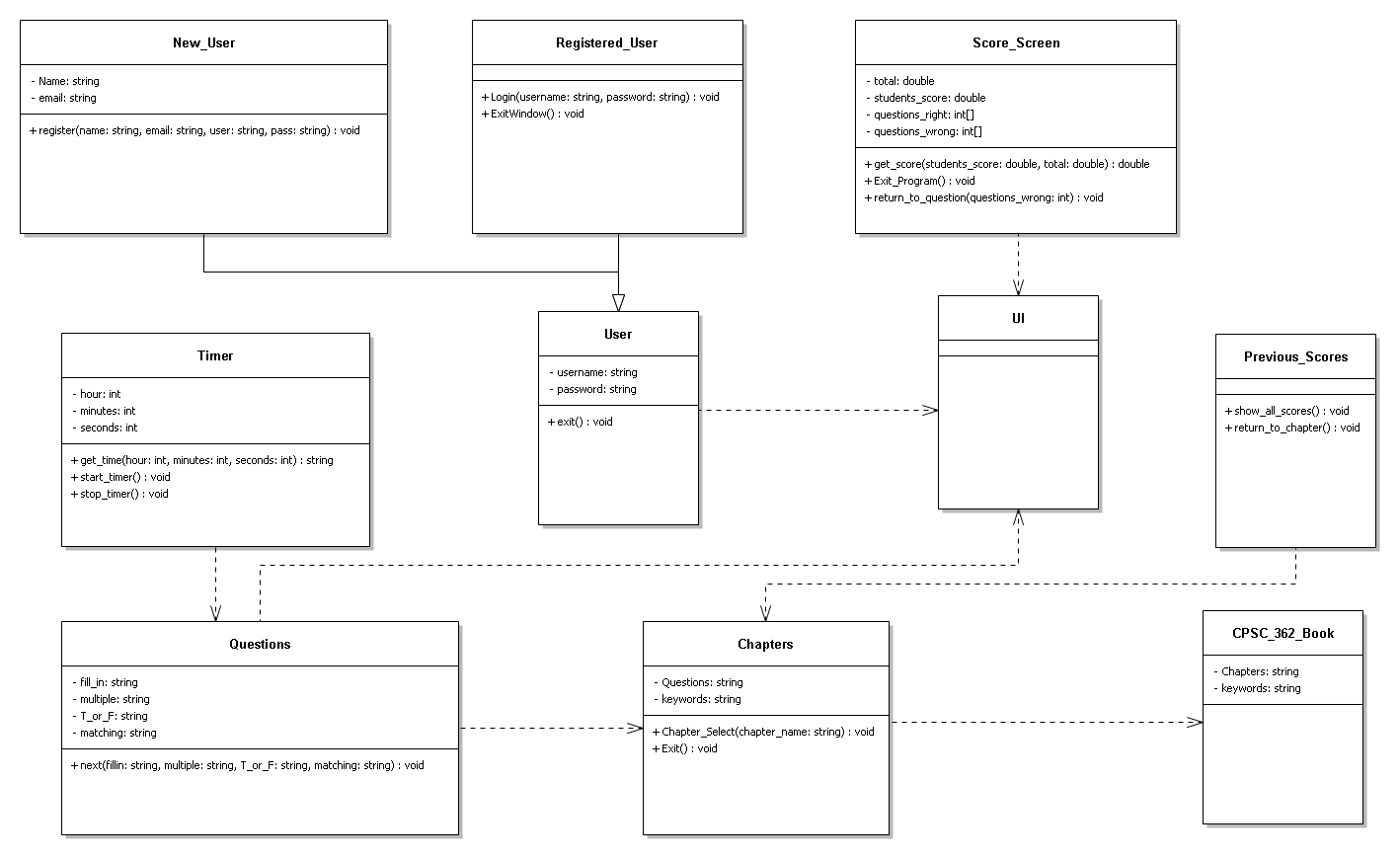




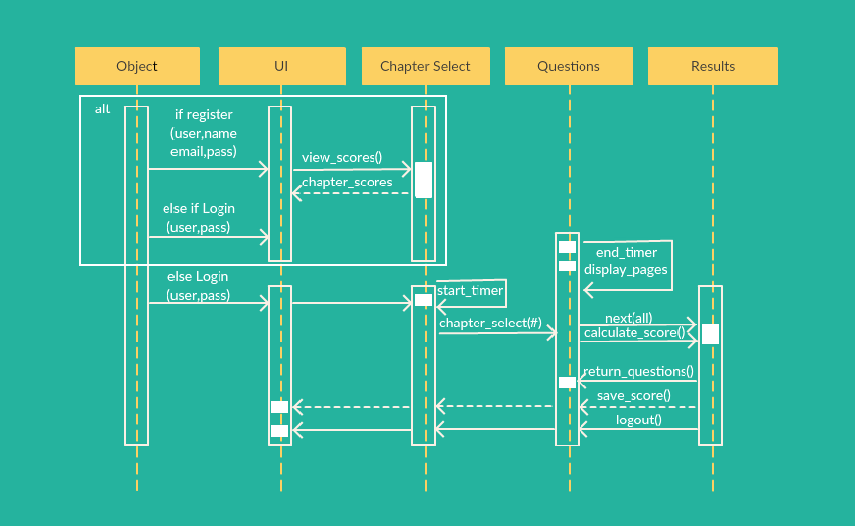
**Explain how your team developed your system and documented activities by producing work products related to this phase.**

This is our second to last iteration before the final release. Therefore the only feature we implemented was having a separate results page for the user. The user would be able to see a certain amount of past scores they got when taking the quiz. Where the scores would be saved in a type of database. This was integrated into our program by the developers. Which were then tested by the QA to see if the scores properly saved. This was all aided by the use of user stories, and test cases.

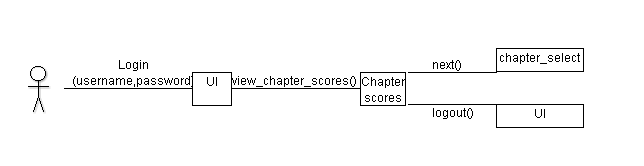
**7. Class diagram**



**8. System Sequence Diagram**



**9. Communication Diagrams based on your Class diagram.**



**10. Test Plan, Test suite and Test Cases.**

**Iteration 3 Test Cases**

|  |  |
| --- | --- |
| **Test Designed by: David Tran** | **Module Name: Chapter quiz page with timer** |
| **Test Designed date: 4-24-16** | **Test Title: Chapter quiz with a timer set to beep** |
| **Test Case ID: 27** | **Description: During the chapter quiz, there is a timer that would beep at the last 10 seconds** |
| **Test Priority (Low/Medium/High): High** | **Pre-conditions: User is logged in and is on the chapter menu** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Step** | **Test Steps** | **Test Data** | **Expected Result** | **Actual Result** | **Status (Pass/Fail)** | **Notes** |
| **1** | **User selects the chapter to take a quiz on.** |  | **The quiz loads for that specific chapter and a timer would start.** | **The quiz loads for that specific chapter and a timer would start.** | **Pass** |  |
| **2** | **User proceeds to take the quiz** |  | **The timer would beep at the last 10 seconds of the quiz.** | **The timer would beep at the last 10 seconds of the quiz.** | **Pass** |  |

**Post-conditions:**

**User is finished with the chapter’s quiz**

|  |  |
| --- | --- |
| **Test Designed by: David Tran** | **Module Name: Chapter quiz’s timer flashing** |
| **Test Designed date: 4-24-16** | **Test Title: Chapter quiz with a timer set to beep and flash red at the same time** |
| **Test Case ID: 28** | **Description: During the chapter quiz, there is a timer that would beep at the last 10 seconds. The timer would also starting flashing red as a visual indicator that time is almost up.** |
| **Test Priority (Low/Medium/High): High** | **Pre-conditions: User is logged in and is currently taking a chapter quiz** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Step** | **Test Steps** | **Test Data** | **Expected Result** | **Actual Result** | **Status (Pass/Fail)** | **Notes** |
| **1** | **User proceeds to take the quiz** |  | **The timer would beep at the last 10 seconds of the quiz. When it beeps, the timer starts flashing red.** | **The timer would beep at the last 10 seconds of the quiz. When it beeps, the timer starts flashing red.** | **Pass** |  |

**Post-conditions:**

**User is finished with the chapter’s quiz**

|  |  |
| --- | --- |
| **Test Designed by: David Tran** | **Module Name: Review button on Chapter Menu page** |
| **Test Designed date: 4-24-16** | **Test Title: View past chapter quiz scores in the review page** |
| **Test Case ID: 29** | **Description: The review button page will allow the user to view their past scores for the quizzes that they have taken.** |
| **Test Priority (Low/Medium/High): High** | **Pre-conditions: User is logged in and is on the chapter menu** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Step** | **Test Steps** | **Test Data** | **Expected Result** | **Actual Result** | **Status (Pass/Fail)** | **Notes** |
| **1** | **User clicks on the review button.** |  | **User is able to view their past quiz scores.** | **User is able to view their past quiz scores.** | **Pass** |  |

**Post-conditions:**

**User is finished with the program.**

**11. 2 Unit Tests**

1. **TestTick Function**

**Test**

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using unitTestTestTick;

using Test;

namespace UnitTestProject1

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestMethod1()

{

System.DateTime endTime;

endTime = DateTime.Now;

Test.testtick = new Test();

string result = Function.testtick(endTime);

string checkEndTime = endTime.ToString();

Assert.AreEqual(result, checkEndTime);

}

}

}

**Function**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Linq;

using System.Text;

using System.IO;

namespace UnitTestProject1

{

class Test

{

System.DateTime startTime;

private String testick(DateTime endTime)

{

TimeSpan endoftime = (endTime - startTime);

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}",

endoftime.Hours, endoftime.Minutes, endoftime.Seconds,

endoftime.Milliseconds / 10);

return elapsedTime;

}

}

}

**2. Results Function**

**Test**

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Test;

//Contains the class for the Unit Test

namespace UnitTestResults

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestMethod1()

{

string[] arrayC = new string[13] {"","correct","incorrect","correct","incorrect", "incorrect", "incorrect", "incorrect",

"incorrect","incorrect","incorrect","incorrect","incorrect",};

string[] returnf= new string[13] {"","true","false","true","aaa","agile","scrum","a","b","c","b","c","a"};

//Test.Result = new Test();

Test.Result = new Test();

bool result = Test.Result(2, arrayC, 5, returnf, "2:25", "stephenchan", 3);

Assert.AreEqual(result, true);

}

}

}

**Function**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using VerifyingResults;

using UnitTest1;

//This is the function for the results.

namespace Test1

{

public class Test

{

public bool Result(int \_correct, string[] arrayC, int c, string[] returnf, string t, string user, int count)

{

InitializeComponent();

check = count;

name = user;

correction = \_correct;

//Calculate the percentage and round up to two decimal

calculation = Convert.ToDouble(correction / Convert.ToDouble(12));

calculation = calculation \* 100;

calculation = Math.Round(calculation, 2);

chapter = c;

total = \_correct.ToString();

Percentage = calculation.ToString();

lbltotal.Text = total + "/12";

lblPercent.Text = Percentage + "%";

time = t;

lbltime.Text = time;

//A switch case to know which questions are from so it can read the page number from the

//text file of the chapter

switch (chapter)

{

case 3:

pagenumber = System.IO.File.ReadAllLines("chapter3.txt");

break;

case 4:

pagenumber = System.IO.File.ReadAllLines("chapter4.txt");

break;

case 5:

pagenumber = System.IO.File.ReadAllLines("chapter5.txt");

break;

case 8:

pagenumber = System.IO.File.ReadAllLines("chapter8.txt");

break;

case 9:

pagenumber = System.IO.File.ReadAllLines("chapter9.txt");

break;

case 10:

pagenumber = System.IO.File.ReadAllLines("chapter10.txt");

break;

}

//Output the Question number and if the user answered correct or incorrect

//on the data grid.

for (int i = 1; i <= 12; ++i)

{

ResultGrid.Rows.Add("Question " + i, arrayC[i], pagenumber[i]);

give\_back[i] = returnf[i];

}

//This for loop change the text color of "Incorrect" to red and "Correct" to

//green in the datagridview

for (int i = 0; i < 12; ++i)

{

if (ResultGrid.Rows[i].Cells[1].Value.ToString() == "Incorrect")

ResultGrid.Rows[i].Cells[1].Style = new DataGridViewCellStyle { ForeColor = Color.Red };

else

ResultGrid.Rows[i].Cells[1].Style = new DataGridViewCellStyle { ForeColor = Color.Green };

}

//Don't let the Result datagrid to sort

for (int i = 0; i < 3; ++i)

{

ResultGrid.Columns[i].SortMode = DataGridViewColumnSortMode.NotSortable;

}

if (check == 0)

{

//Will write the new score into the score.txt score = System.IO.File.ReadAllLines("score.txt");

//Find how many in the textfile

string line = "";

StreamReader sr = new StreamReader("new.txt");

int counter = 0;

while ((line = sr.ReadLine()) != null)

{

++counter;

}

score = new String[counter];

sr.Close();

//Copy every line into array

StreamReader sr2 = new StreamReader("new.txt");

int counter2 = 0;

while ((line = sr2.ReadLine()) != null)

{

score[counter2] = line;

++counter2;

}

sr2.Close();

people\_list = score.Count();

//change the chapter from int to string

string chp = c.ToString();

//change the number of correct to string

string chapter\_score = \_correct.ToString();

//This function add new name, chapter or score to the text file contain

//the record of the user's testing score

for (int i = 0; i < people\_list; ++i)

{

string[] split = score[i].Split(separators, StringSplitOptions.RemoveEmptyEntries);

//The first value in word[] is the username, so when the username is match to the

//username inputted from the login screen, then the datagrid will output the user's

//past test score

if (split[0] == user)

{

string new\_score = chp + "." + chapter\_score + "/";

score[i] = score[i].Replace(user + "/", user + "/" + new\_score);

break;

}

//If the user doesn't have score in text file containing the score (score.txt) then add the name, chapter and score

else if (i == people\_list - 1 && split[0] != user)

{

new\_tester = user + "/" + chp + "." + chapter\_score + "/";

break;

}

}

sr.Close();

//This would write back all the name, chapter and score back to the text file

StreamWriter sw = new StreamWriter("new.txt");

for (int i = 0; i < score.Length; ++i)

{

sw.WriteLine(score[i]);

}

//If the user have not yet take the test, then this function will write the user

//name, chapter and score in the text file

if (new\_tester != "")

{

sw.WriteLine(new\_tester);

}

sw.Close();

return true;

}

}

}

}

**12. Automation For Iteration 2 Test Cases**

**Test Case: Viewing Colored Text**

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows.Input;

using System.Windows.Forms;

using System.Drawing;

using Microsoft.VisualStudio.TestTools.UITesting;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Microsoft.VisualStudio.TestTools.UITest.Extension;

using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;

namespace AutomationTest1

{

/// <summary>

/// Summary description for CodedUITest1

/// </summary>

[CodedUITest]

public class CodedUITest1

{

public CodedUITest1()

{

}

[TestMethod]

public void CodedUITestMethod1()

{

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

this.UIMap.coloredText()

}

#region Additional test attributes

// You can use the following additional attributes as you write your tests:

////Use TestInitialize to run code before running each test

//[TestInitialize()]

//public void MyTestInitialize()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

////Use TestCleanup to run code after each test has run

//[TestCleanup()]

//public void MyTestCleanup()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

#endregion

/// <summary>

///Gets or sets the test context which provides

///information about and functionality for the current test run.

///</summary>

public TestContext TestContext

{

get

{

return testContextInstance;

}

set

{

testContextInstance = value;

}

}

private TestContext testContextInstance;

public UIMap UIMap

{

get

{

if ((this.map == null))

{

this.map = new UIMap();

}

return this.map;

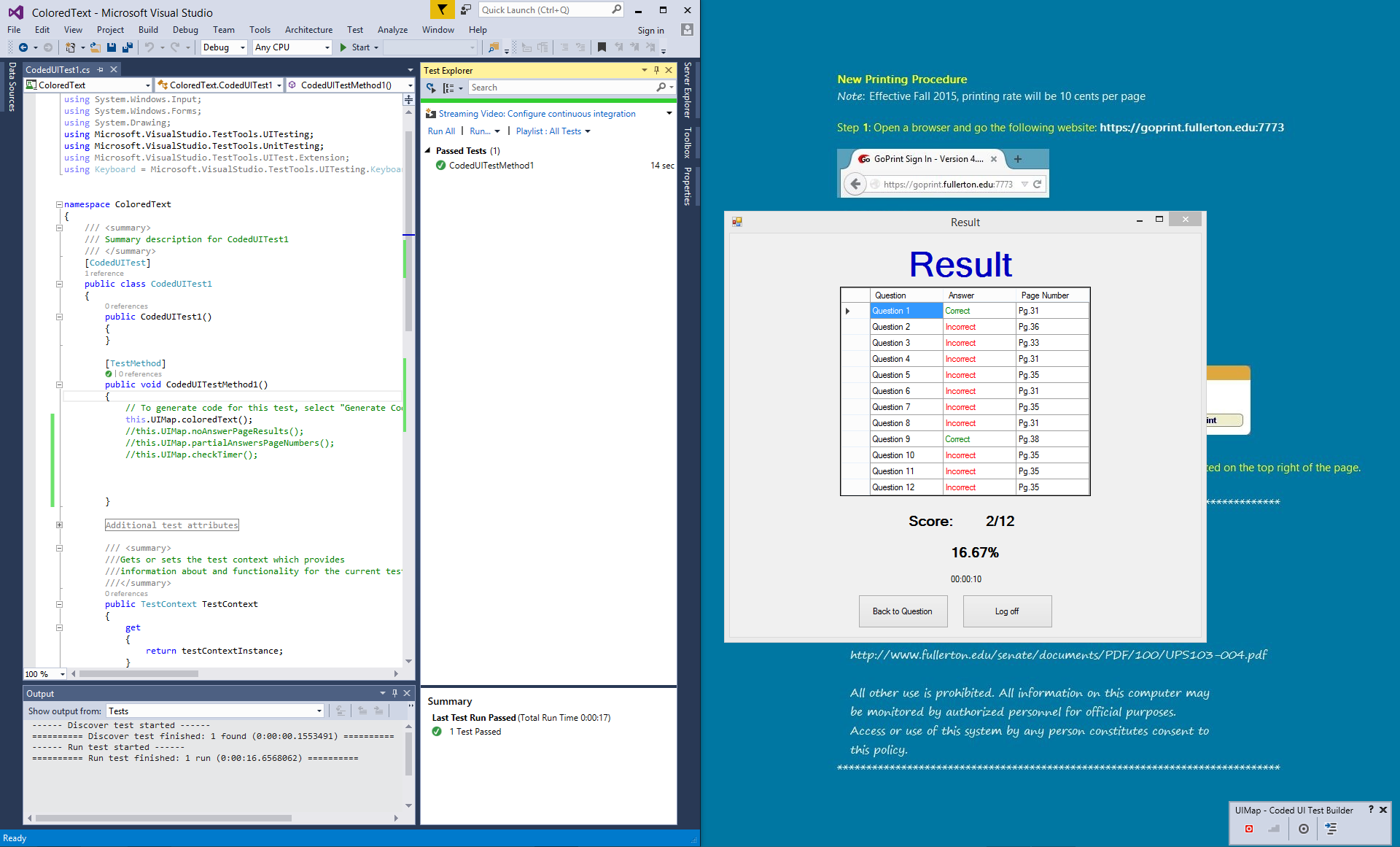
}

}

private UIMap map;

}

}



**Test Case: No Answer Page Results**

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows.Input;

using System.Windows.Forms;

using System.Drawing;

using Microsoft.VisualStudio.TestTools.UITesting;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Microsoft.VisualStudio.TestTools.UITest.Extension;

using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;

namespace AutomationTest3

{

/// <summary>

/// Summary description for CodedUITest1

/// </summary>

[CodedUITest]

public class CodedUITest1

{

public CodedUITest1()

{

}

[TestMethod]

public void CodedUITestMethod1()

{

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

this.UIMap.noAnswerPageResults();

}

#region Additional test attributes

// You can use the following additional attributes as you write your tests:

////Use TestInitialize to run code before running each test

//[TestInitialize()]

//public void MyTestInitialize()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

////Use TestCleanup to run code after each test has run

//[TestCleanup()]

//public void MyTestCleanup()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

#endregion

/// <summary>

///Gets or sets the test context which provides

///information about and functionality for the current test run.

///</summary>

public TestContext TestContext

{

get

{

return testContextInstance;

}

set

{

testContextInstance = value;

}

}

private TestContext testContextInstance;

public UIMap UIMap

{

get

{

if ((this.map == null))

{

this.map = new UIMap();

}

return this.map;

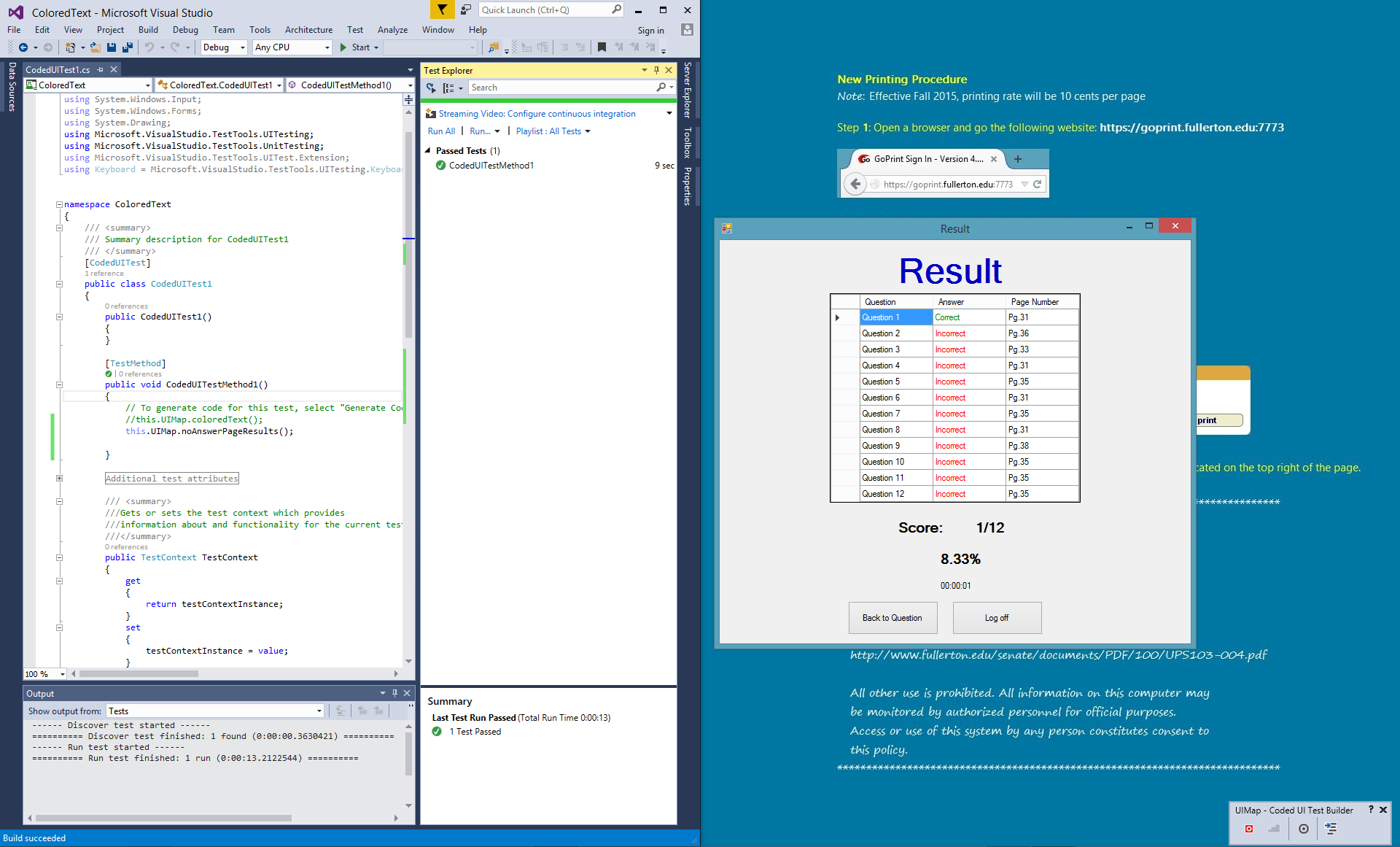
}

}

private UIMap map;

}

}



**Test Case: Partial Answers Page numbers**

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows.Input;

using System.Windows.Forms;

using System.Drawing;

using Microsoft.VisualStudio.TestTools.UITesting;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Microsoft.VisualStudio.TestTools.UITest.Extension;

using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;

namespace AutomationTest1

{

/// <summary>

/// Summary description for CodedUITest1

/// </summary>

[CodedUITest]

public class CodedUITest1

{

public CodedUITest1()

{

}

[TestMethod]

public void CodedUITestMethod1()

{

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

this.UIMap.partialAnswersPageNumbers();

}

#region Additional test attributes

// You can use the following additional attributes as you write your tests:

////Use TestInitialize to run code before running each test

//[TestInitialize()]

//public void MyTestInitialize()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

////Use TestCleanup to run code after each test has run

//[TestCleanup()]

//public void MyTestCleanup()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

#endregion

/// <summary>

///Gets or sets the test context which provides

///information about and functionality for the current test run.

///</summary>

public TestContext TestContext

{

get

{

return testContextInstance;

}

set

{

testContextInstance = value;

}

}

private TestContext testContextInstance;

public UIMap UIMap

{

get

{

if ((this.map == null))

{

this.map = new UIMap();

}

return this.map;

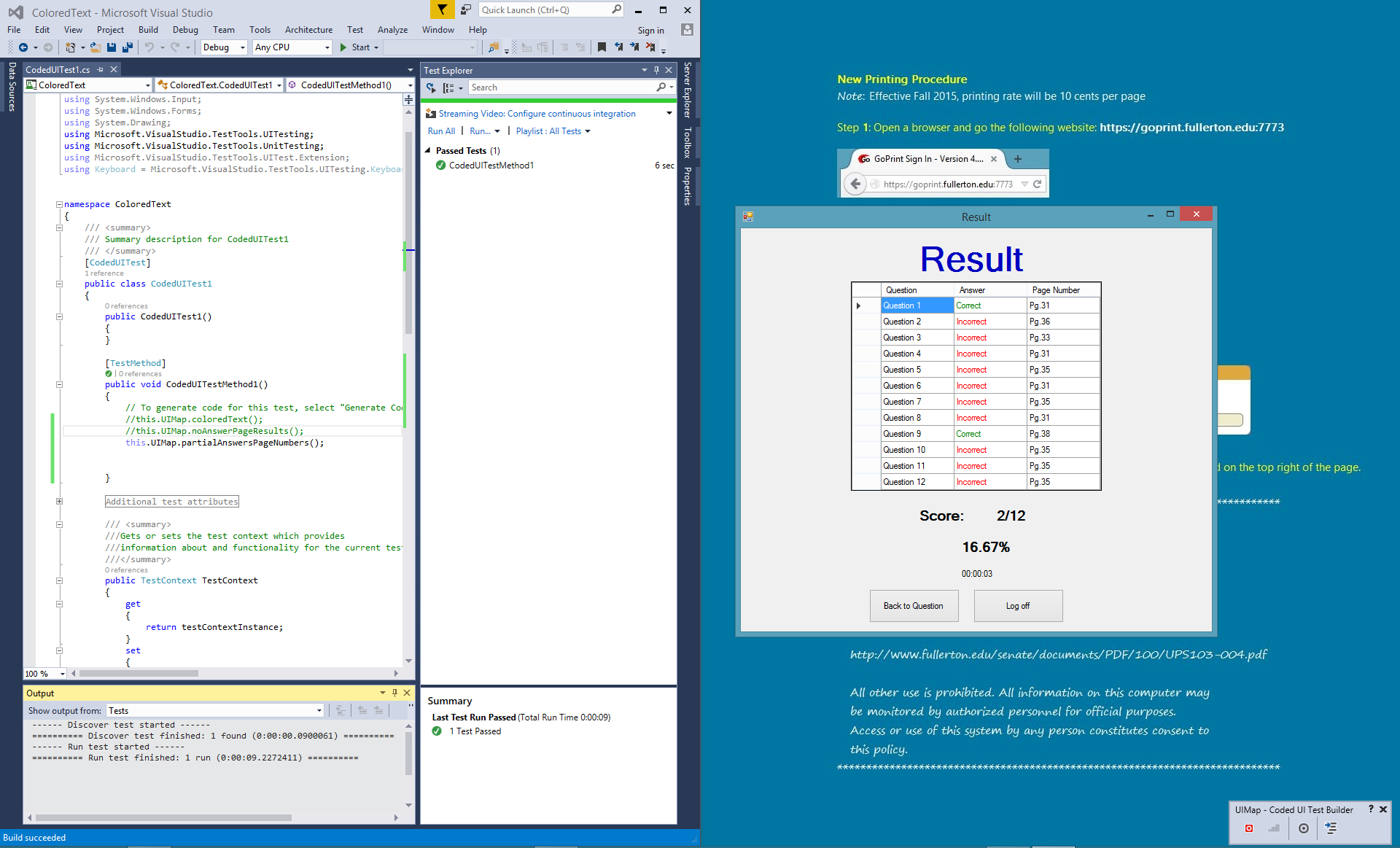
}

}

private UIMap map;

}

}



**Test Case: Check Timer**

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows.Input;

using System.Windows.Forms;

using System.Drawing;

using Microsoft.VisualStudio.TestTools.UITesting;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Microsoft.VisualStudio.TestTools.UITest.Extension;

using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;

namespace AutomationTest1

{

/// <summary>

/// Summary description for CodedUITest1

/// </summary>

[CodedUITest]

public class CodedUITest1

{

public CodedUITest1()

{

}

[TestMethod]

public void CodedUITestMethod1()

{

// To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

this.UIMap.checkTimer();

}

#region Additional test attributes

// You can use the following additional attributes as you write your tests:

////Use TestInitialize to run code before running each test

//[TestInitialize()]

//public void MyTestInitialize()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

////Use TestCleanup to run code after each test has run

//[TestCleanup()]

//public void MyTestCleanup()

//{

// // To generate code for this test, select "Generate Code for Coded UI Test" from the shortcut menu and select one of the menu items.

//}

#endregion

/// <summary>

///Gets or sets the test context which provides

///information about and functionality for the current test run.

///</summary>

public TestContext TestContext

{

get

{

return testContextInstance;

}

set

{

testContextInstance = value;

}

}

private TestContext testContextInstance;

public UIMap UIMap

{

get

{

if ((this.map == null))

{

this.map = new UIMap();

}

return this.map;

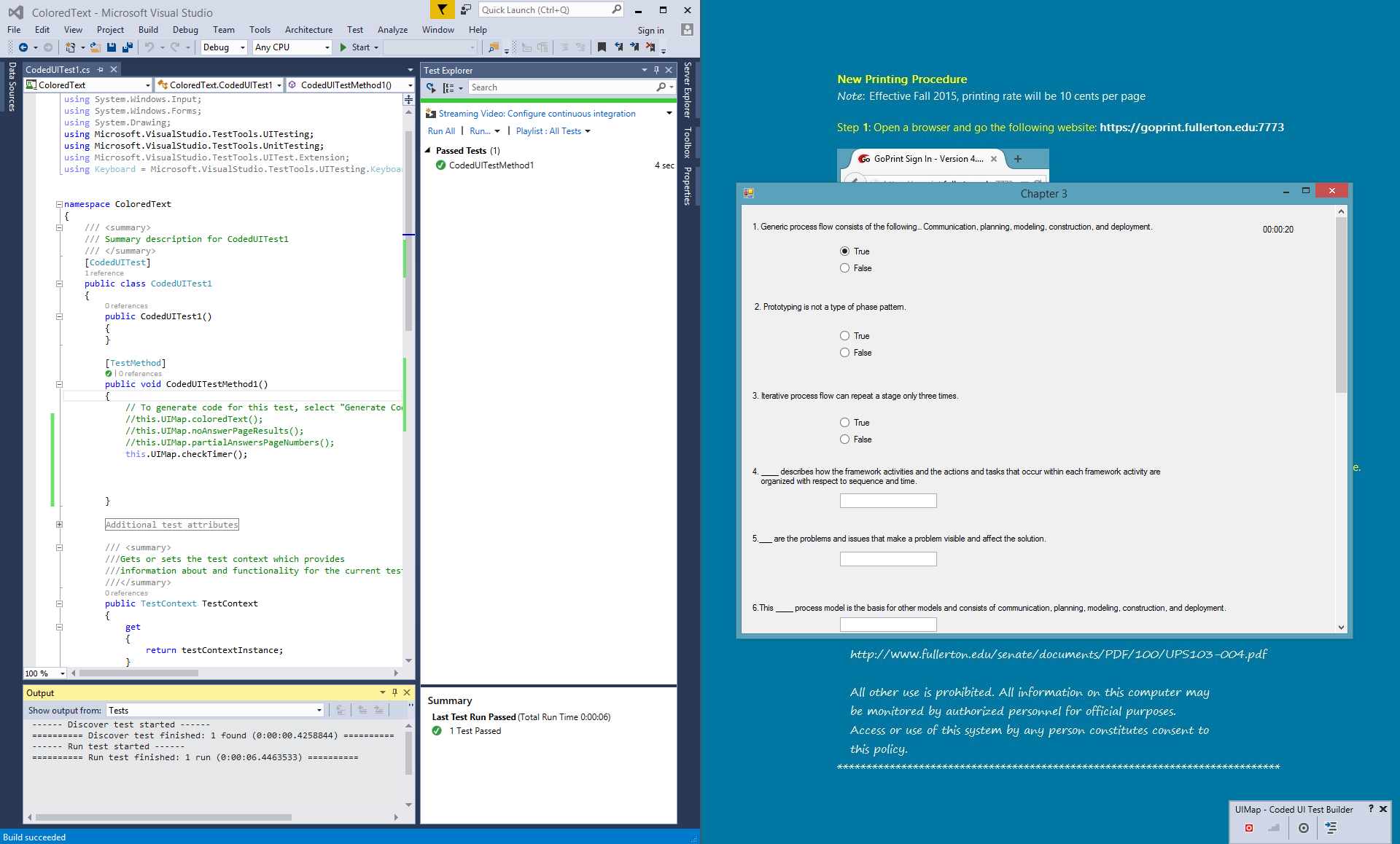
}

}

private UIMap map;

}

}



**13. Your code for the 1 functional user story.**

**A. Saving User Chapter Score**

//The check was used as a counter when user return back to Result Screen

//after viewing the correct answer, the program will not add the score again.

if (check == 0)

{

//Will write the new score into the score.txt score = System.IO.File.ReadAllLines("score.txt");

//Find how many in the textfile

string line = "";

StreamReader sr = new StreamReader("score.txt");

int counter = 0;

while ((line = sr.ReadLine()) != null)

{

++counter;

}

score = new String[counter];

sr.Close();

//Copy every line into array

StreamReader sr2 = new StreamReader("score.txt");

int counter2 = 0;

while ((line = sr2.ReadLine()) != null)

{

score[counter2] = line;

++counter2;

}

sr2.Close();

people\_list = score.Count();

//change the chapter from int to string

string chp = c.ToString();

//change the number of correct to string

string chapter\_score = \_correct.ToString();

//This check if there are any users in the text file

if (people\_list > 0)

{

//This function add new name, chapter or score to the text file contain

//the record of the user's testing score

for (int i = 0; i < people\_list; ++i)

{

string[] split = score[i].Split(separators, StringSplitOptions.RemoveEmptyEntries);

//The first value in word[] is the username, so when the username is match to the

//username inputted from the login screen, then the datagrid will output the user's

//past test score

if (split[0] == user)

{

string new\_score = chp + "." + chapter\_score + "." + InputDate + "|";

score[i] = score[i].Replace(user + "|", user + "|" + new\_score);

break;

}

//If the user doesn't have score in text file containing the score (score.txt) then add the name, chapter and score

else if (i == people\_list - 1 && split[0] != user)

{

new\_tester = user + "|" + chp + "." + chapter\_score + "." + InputDate + "|";

break;

}

}

}

//If there are no users in the textfile just add it in

else

{

new\_tester = user + "|" + chp + "." + chapter\_score + "." + InputDate + "|";

}

sr.Close();

//This would write back all the name, chapter and score back to the text file

StreamWriter sw = new StreamWriter("score.txt");

for (int i = 0; i < score.Length; ++i)

{

sw.WriteLine(score[i]);

}

//If the user have not yet take the test, then this function will write the user

//name, chapter and score in the text file

if (new\_tester != "")

{

sw.WriteLine(new\_tester);

}

sw.Close();

}

**B. Reading User Chapter Score**

//Read all the users and their name into list

StreamReader sr = new StreamReader("score.txt");

while ((line = sr.ReadLine()) != null)

{

list.Add(line);

}

//count number of index in list

people\_list=list.Count();

for (int i = 0; i < people\_list; ++i)

{

//In the word array, it will contain a username and his or her score

string[] words = list[i].Split(separators, StringSplitOptions.RemoveEmptyEntries);

//The first value in word[] is the username, so when the username is match to the

//username inputted from the login screen, then the datagrid will output the user's past test score

if (words[0] == user)

{

//This count how many chapter scores are in the words array

int m = words.Count();

//Input the chapter and the chapter score into the DataGrid

for (int j = 1; j < m; ++j)

{

ch = words[j];

++j;

chp\_score = words[j];

++j;

date\_taken = words[j];

TotalGrid.Rows.Add(ch, chp\_score, date\_taken);

}

}

}

sr.Close();

**14. User manual – Screen shots of your working product with explanation on How to use your system.**

**Registration Screen**



1. If you are not a member, then

1. Enter your full name

2. Enter your E-mail

3. Enter a username

a. Username cannot be the same as existed one

4. Enter a password

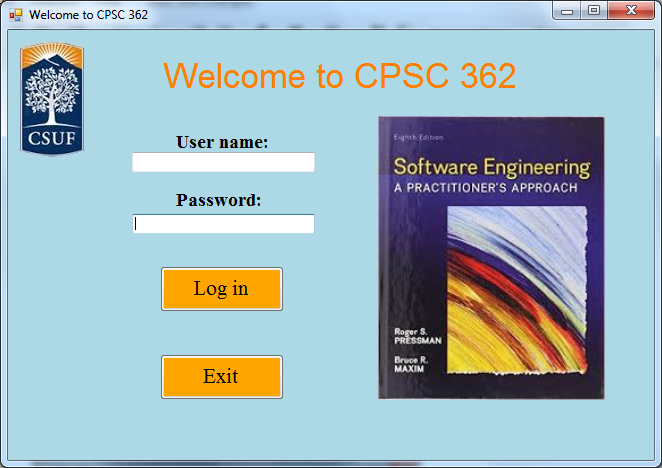
a. Need to be more than 8 characters

5. Click on the “Register” button to become a member

2. If you are a member, then

1. Click on the “Member Login” button to go to Login screen

**Login Screen**



If users want to go to Chapter Menu Screen

1. Enter your User name

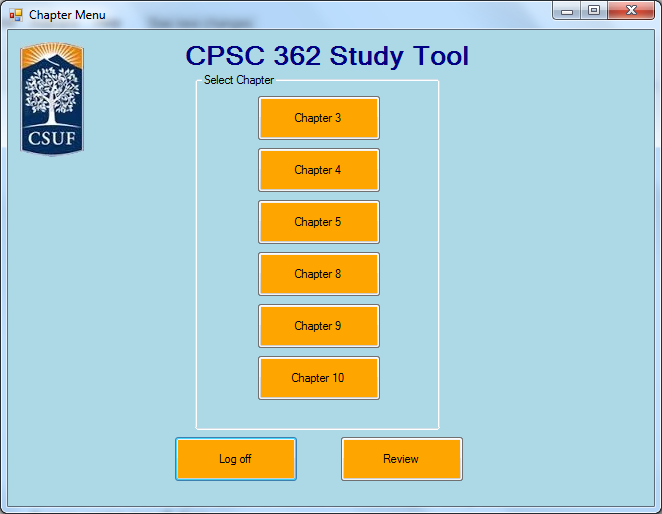
2. Enter your Password

3. Click on “Log in” button

If users want to exit the program, then

1. Click on the “Exit” button

**Chapter Menu Screen**



If users want to start answer the questions on study tool,

1. Click one of the chapter button to start the selected chapter study tool

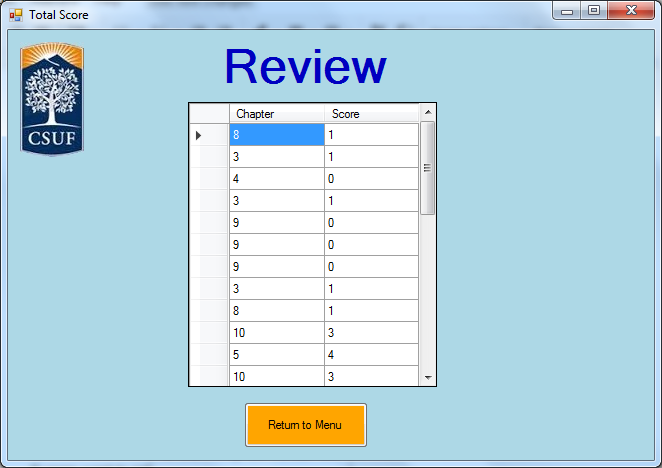
If users want to log off, then

1. Click the “Log off” button to log off the account.

If users want to see all their scores that they have gotten before, then

1. Click the “Review” button

**Review Screen**

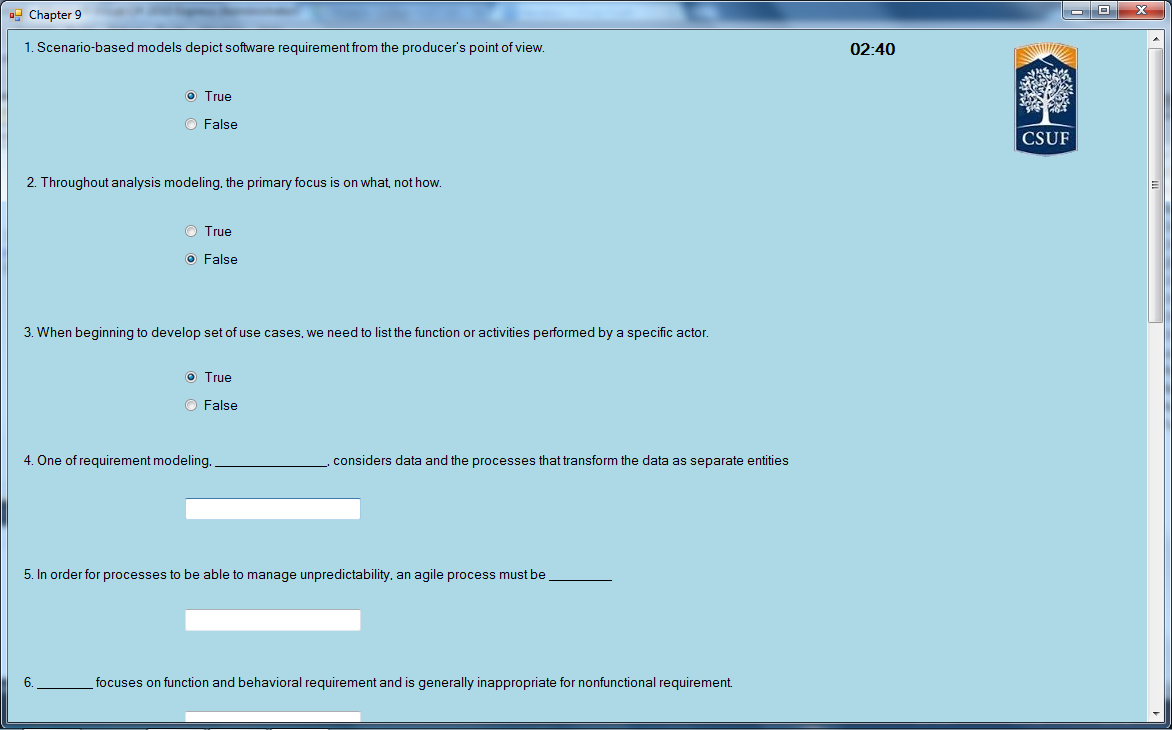


1. The users can see all their previous scores taken from the newest to oldest.

If users want to return to the Chapter Screen, then

1. Click the “Return to Menu” button

**Chapter Screen**



1. The users will be given 25 minutes to answer the given questions.

2. When the timer reaches 24:50, there will be beeping sounds, and the timer will start flashing red to signal users that they only have 10 seconds left.

3. If the timer reaches 25 minutes, then it will automatically take the users to the Result Screen.

4. Click “Finish” button to submit the answers and go to Result Screen

5. Users can click “Back to menu” button to go back to Chapter Menu Screen

**Result Screen**



1. Users can see if they answered their questions correctly and what page they can find the answers to the question. They can see the score and percentage they got for the current chapter they are answering.

2. Users can click on “Back to Question” button to go back to the chapter questions screen with answers shown.

3. Users can click on “Log off” button to log off their account and go to the log in screen.

**15. References (list references here, and cite them in appropriate places in the report)**

Software Engineering: A Practitioner’s Approach by Roger Pressman

**16. Team Charter (in the given format)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Course Title** | CPSC 362 |  | All team members participated in the creation of this charter and agree with its content. **Date** 01/28/2014 |
| **Instructor** | Yasamin Ehteshami |  |
| **Course Dates** | MW 5:00-6:50 |  |

**Team Members** (Contact Information)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Address (city, state, country) | Phone | Cell | Email |
| Stephen Chan | La Mirada, CA | 562-650-2700 |  | stephenchan@csu.fullerton.edu |
| David Tran | Anaheim, CA | 714-487-2653 |  | DavidTran794@csu.fullerton.edu |
| Michael Ha | Westminster, CA | 714-725-9569 |  | Mha94@csu.fullerton.edu |
| Kevin Le | Westminster, CA | 714-925-3210 |  | lekevin42@csu.fullerton.edu |
| Jonathan Peng | Chino Hills, CA | 909-348-4201 |  | jspeng@csu.fullerton.edu |
| David Luong | Santa Ana, CA | 714-360-3083 |  | davidluong@csu.fullerton.edu |

**Team Member Skill Inventory** (Areas individual members can contribute)

|  |  |
| --- | --- |
| Stephen Chan | § C++,SQL,Python |
| David Tran | § C++, Python, HTML5, SQL |
| Michael Ha | § C++, Python, Assembly |
| Kevin Le | § C++, python, C, Assembly |
| Jonathan Peng | § C++,Assembly, Java, C# |
| David Luong | § C++, Java, Python, CSS, HTML server management and hosting experience. |

**Team Goals** (Project goals, team process goals, quality goals, etc.)

|  |
| --- |
| § Efficient Application, Finish a few days before deadline. Program will compile without errors. |

**Team Roles** (Define roles of each member to achieve goals)

|  |  |
| --- | --- |
| Stephen Chan | § Scrum Master, Recorder, Developer |
| David Tran | § Developer, Tester |
| Michael Ha | § Developer, tester |
| Kevin Le | § Developer, tester |
| Jonathan Peng | § Developer, tester |
| David Luong | § Developer, Tester |

**Ground Rules** (Meeting schedule/locations, attendance expectations, agenda, assignment completion, communication methods, etc.)

|  |
| --- |
| § Group will meet and discuss project in Google Hangout sessions.  § All team members must be punctual and prepared for each team meeting.  § Participation and input is expected from all team members. All opinions will be considered and equally valued.  § The team will meet at least once each week via chat or conference call to discuss current and upcoming projects or assignments (tentatively scheduled for every Saturday @ 8:30 pm).  § Team members will notify the lead in advance if they are not going to be able to attend a scheduled meeting.  § Team members should check email at least once a day to stay on top of things.  § Team members should reply to email within 24 hours.  § Team members will turn in team assignments no later than two weeks prior to the due date.  § All team members will be held accountable for their portions of the projects and are expected to complete them in a timely manner and doing the best job they can.  § Notify team of emergencies that may result in not being able to meet deadlines or meetings. The rest of the team will do their best to pitch in on the team assignment.  § The team must maintain open, clear, and effective communication at all times.  § Assist fellow team members when they are in need.  Team will collectively decide when to meet.  Team will not form alliances or teams against one another.  § Maintain a positive, honest, and open atmosphere by respecting other members’ suggestions, using constructive criticism, and encouragement. |

**Time Commitments/Availability** (Pacific Time)

|  |  |
| --- | --- |
| Stephen Chan | § Friday ALL DAY, Weekend(free) |
| David Tran | § Friday(afternoon-night), Sunday(Morning,Night) |
| Michael Ha | § All day fri-sun, mon-thurs anytime except 5-9pm |
| Kevin Le | § Friday - Sunday |
| Jonathan Peng | § Weekend(free) |
| David Luong | § Everyday(morning) |

**Conflict Management** (What are potential conflicts that might arise among or between team members during this course? How will team members deal with these and other conflicts?)

|  |
| --- |
| Scrum Master will decide course of action.  § In order to avoid conflict clear roles and responsibilities must be assigned, so that there is no confusion.  § If a team member is not performing, the team lead will speak to the member and try and resolve the issue.  § If conflicts arise, please bring them up to the whole team so that everybody can help to resolve the issue in a peaceful and harmonious manner.  § All team members must settle conflicts within the group as quickly as possible. |

**Risk Management** (What are potential barriers to the achievement of these goals?)

|  |
| --- |
| Project Must be in a programming language all developers are comfortable with.  § Scrum Master and Recorder will be in charge of managing  § Any issues between the Scrum Master and the Recorder will be resolved by involving the entire group  § List risks that are chances or possibilities of suffering loss or danger in the project.   * Computer breaks.   + Solution: Make sure to backup files on USB or email * Files are lost for any reason.   + Solution: Upload to Google Drive. * The possibility that we will not finish project on time.   + Solution: Make sure to keep an eye on progress made throughout the semester, set deadlines * Arguments that threaten the group project   + Solution: Make sure to address everyone’s concerns before moving forward |

**Team Evaluation Criteria** (List evaluation criteria that will be used to evaluate team members objectively.)

|  |
| --- |
| * Response time for emails, texts. * Updates on current progress of assignments. * Whether they’re able to attend meetings. * Actual completion of project components. * Evaluate performance based upon efficient code and time of submission. |

**17. Team Evaluation (in the given format).**