**Object-Oriented Software Development Project**

**A Hands-On Approach**

**Online Registration System of the University**

**By Team 9**

**Phase V: System Integration and Testing**

**Table of Contents**

1. **Test Schedule**
   * 1. Test Schedule
     2. Function Tests
     3. Performance Tests
     4. Stress Tests
     5. Acceptance Test (Alpha Test)
     6. Installation Test (Beta Test)
2. **Test Analysis**
3. **Tested and Documented Program Listings**
4. **Gantt Chart**
5. **Reference**

**System Integration and Testing**

1. **Test Schedule**
   * 1. **Test Schedule**

Test:

Function Test:

Performance Test:

Stress Test:

Acceptance Test:

Installation Test:

Test Date:

Tuesday, April 4, 2017

Tuesday, April 4, 2017

Tuesday, April 4, 2017

Thursday, April 6, 2017

Thursday, April 6, 2017

Tester:

Nicholas Mallonee

Carson Schutter

Mi Gao

Marcus Fields

Eric O’Sullivan

1. **Function Tests**

The Function testing is A functional test is used to test the various operations of the system to ensure it is working as intended. Throughout the development of the system we will be monitoring and testing each function, such as adding a class or dropping one, to ensure it works. The original part of the software implementation part is to allow students to log in and register or delete the course. Based on the algorithms we intend to use, the implemented system meets these requirements, as tested by Nicholas Mallonee.

1. **Performance Tests**

The performance testing is supposed to test the speed and effectiveness of a program. We are testing our software by measuring how long it takes to load and perform the given tasks of registration. Based on the algorithms we intend to use, the system works efficiently, securely, accurately, and reliably. The performance was cross checked with the requirement specification document and was found to be well implemented from the performance side.

1. **Stress Tests**

Stress testing involves testing the system to identify the boundaries and its limitations. The system is typically tested with abnormally high demands or users to see how the system handles the overload. This could not be tested due to the software system not being in a public domain where such a measurement would be allowed. However, adding multiple sections to the system to test the limitations of the number of sections allowed tested the system. This limitation was not found yet.

1. **Acceptance Test (Alpha Test)**

The system characteristics were analyzed to test if they are in compliance with the defined system requirements that were specified at the beginning of the project. Due to time constraints and feasibility, the system interface changes slightly, but most are still accurate to the previously specified standards. The test was performed and found to be quite accurate with the specified criteria.

1. **Installation Test (Beta Test)**

The installation test is used to perform system functions to allow additional errors that may be missing. The testers were tested with other teammates. They exercised the system function, found that the implementation of the software system in accordance with the requirements of the system specification file implementation and do not find other errors yet.

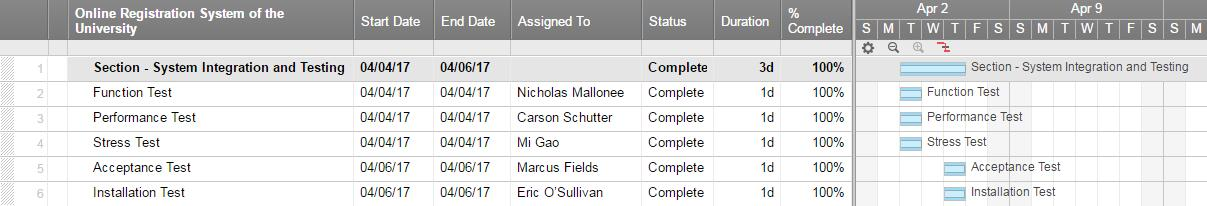
1. **Test Analysis**

For now, the subsystem we are concerned with is the UI for the user portal window. We asked to log in to check whether the input credentials are in the current user's database. The login framework is doing this, and the message user's class conforms to our different requirements, and demonstrates what needs to be done to make the system complete. Moreover, in order to improve the efficiency of the software, we simulate a student logging on rapidly and test the ability of the system to handle many requests for date, adding or dropping classes, and logging out. We are currently looking into automating this test to further increase the speed and stress we can apply to the system.

1. **Tested and Documented Program Listings**

See later pages of the appendix.

1. **Gantt Chart**

****

1. **Reference**

*Lee, R., 2013. Software Engineering: A Hands-On Approach. Atlantis Press, Bücher.*

*Qt UI Design. https://www.qt.io/ui/. The Qt Company.*

**Algorithms.cpp:**

// For this application we are going to be testing how a user

// responds to the a prediciton done for them. For our tests

// We are going to test to see how the user responsds to a prediciton

// of a couple of simple math problems. For our tests we are going

// to test the user on how they respond to answers that yield to the

// sum of 8.

// We will be reading in a file and use the numbers in there to display

// answer to the end user.

// File Format:

// 6 2 // 6 Will be the X value and 2 will be the Y value.

// 8 0 // 8 Will be the X value and 0 will be the Y Value

// We Will be reading in two lines of the file to get an A and

// B answer.

// Once the answers are read in, then we will randomly pick on of

//the answers. And track if the user has changed their answer. If

// they do not change it we will not track it. Below is the code for all of it.

#include <iostream.h>

int main(void)

{

// Create Varaibles

bool bHasCompletedQuestions = false; // Assume we have not completed the questions

bool bHasChangedAnswer = false; // Assume the user has not changed the answer

bool bHasSelectedOptionA = true; // Assume we choose option A

int numberOfChanges = 0;

int numberOfQuestions = 5;

File\* fileToRead = NULL; // get the file to read from

// If the file exists, then cont.. else return error

if(fileToRead)

{ // While We have not completed the questions

while(!(bHasCompletedQuestions))

{

// Pull Two Lines From the file

// Display them to option A and Option B

// Pick a Random Button to select

// And Track the change if any

bHasSelectedOptionA = getRandomSelection();

// Get any changes if any

// If Change, then increment the changes

numberOfChanges++;

// Reset Variables, and check if we have finished the questions.

}

}

//

// Any Clean Up Needed, close file here ect....

// Display The Results. And Wait to record it

fprintf(strerr, "Number of Changes Made By the User: %d\n", numberOfChanges );

system("pause");

// Return the application thread.

return 0;

}

**main.cpp:**

#include "mainwindow.h"

#include "course.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

// Get the time and date of the Host machine

time\_t currentTime = time(0);

// Create the courses

Course\* classOne = new Course(classID::id\_CPS\_180, Year::Year\_2017, Semester::Sem\_Fall);

Course\* classTwo = new Course(classID::id\_CPS\_181, Year::Year\_2017, Semester::Sem\_Fall);

Course\* classThree = new Course(classID::id\_CPS\_410, Year::Year\_2018, Semester::Sem\_Spring);

Course\* classFour = new Course(classID::id\_CPS\_470, Year::Year\_2018, Semester::Sem\_Spring);

// Generate the Window and the UI and Show it to the end user

QApplication a(argc, argv);

MainWindow w;

// Show the window

w.show();

// Delete the pointers

delete classOne;

delete classTwo;

delete classThree;

delete classFour;

// Return this thread..

return a.exec();

}

**Course.cpp:**

//-----------------------------------------------------------------------------------

// Libraries and Includes

//-----------------------------------------------------------------------------------

#include "course.h"

//-----------------------------------------------------------------------------------

// Constructor -

//-----------------------------------------------------------------------------------

Course::Course(enum classID className, enum Year classyear, enum Semester classterm)

{

classTerm = classterm;

classYear = classyear;

classIDNum = className;

}

//-----------------------------------------------------------------------------------

// Class Mutators -

//-----------------------------------------------------------------------------------

bool Course::setCourseTerm(Semester term)

{

if(term == Semester::Sem\_NULL) // If We do not have a null state

return false;

else

{

if(term != classTerm) // If they are not the same...

{

classTerm = term; // Change the semester term

return true; // Return true that we modified it

}

else // Else if they are the same...

return false; // Return false that we did not modify it

}

}

bool Course::setCourseYear(Year yearOfCourse)

{

if(yearOfCourse == Year::Year\_NULL) // Same Methodology as above...

return false;

else

{

if(yearOfCourse != classYear)

{

classYear = yearOfCourse;

return true;

}

else

return false;

}

}

bool Course::setCourseClassID(classID courseID)

{

if(courseID == classID::id\_NULL) // Same methodology as above

return false;

else

{

if(courseID != classIDNum)

{

classIDNum = courseID;

return true;

}

else

return false;

}

}

//-----------------------------------------------------------------------------------

// Class Accessors -

//-----------------------------------------------------------------------------------

Semester Course::getCourseTerm()

{

return classTerm;

}

Year Course::getCourseYear()

{

return classYear;

}

classID Course::getCourseID()

{

return classIDNum;

}

**course.h:**

#ifndef COURSE\_H

#define COURSE\_H

// -- Enum for Semesters

enum Semester

{

Sem\_Fall,

Sem\_Spring,

Sem\_Summer\_One,

Sem\_Summer\_Two,

Sem\_NULL

};

// -- Enum for Year,

// Note: This is not the best way, but the lightest

// for the static system

enum Year

{

Year\_2017,

Year\_2018,

Year\_2019,

Year\_NULL

};

// -- Enum for class ID

// It would be better to use a string,

// but this makes more sense for a small project

enum classID

{

id\_CPS\_180,

id\_CPS\_181,

id\_ITC\_383,

id\_CPS\_410,

id\_CPS\_450,

id\_CPS\_470,

id\_NULL

};

//#define enum Semester Semester;

//#define enum Year Year;

//#define enum classID classID;

// -- Class Information --

class Course

{

// -- Public Information -- Constructor -- //

public:

Course(enum classID className, enum Year classyear, enum Semester classterm);

// -- Public Information -- Accessors and Mutators for Courses -- //

public:

bool setCourseTerm(enum Semester term = Semester::Sem\_NULL);

bool setCourseYear(enum Year yearOfCourse = Year::Year\_NULL);

bool setCourseClassID(enum classID courseID = classID::id\_NULL);

enum Semester getCourseTerm();

enum Year getCourseYear();

enum classID getCourseID();

// -- Private Information -- Class Information -- //

private:

enum Semester classTerm = Semester::Sem\_NULL;

enum Year classYear = Year::Year\_NULL;

enum classID classIDNum = classID::id\_NULL;

};

#endif // COURSE\_H

**mainwindow.cpp:**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

**mainwindow.h:**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

**mainwindow.ui:**

<?xml version="1.0" encoding="UTF-8"?>

<ui version="4.0">

<class>MainWindow</class>

<widget class="QMainWindow" name="MainWindow">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>723</width>

<height>566</height>

</rect>

</property>

<property name="windowTitle">

<string>CentralMichiganRegistrationForm</string>

</property>

<widget class="QWidget" name="centralWidget"/>

<widget class="QMenuBar" name="menuBar">

<property name="geometry">

<rect>

<x>0</x>

<y>0</y>

<width>723</width>

<height>22</height>

</rect>

</property>

<widget class="QMenu" name="menuClass\_Registration\_Forum">

<property name="title">

<string>Class Registration Forum</string>

</property>

</widget>

<addaction name="menuClass\_Registration\_Forum"/>

</widget>

<widget class="QToolBar" name="mainToolBar">

<attribute name="toolBarArea">

<enum>TopToolBarArea</enum>

</attribute>

<attribute name="toolBarBreak">

<bool>false</bool>

</attribute>

</widget>

<widget class="QStatusBar" name="statusBar"/>

</widget>

<layoutdefault spacing="6" margin="11"/>

<resources/>

<connections/>

</ui>

**QT\_Test\_Project.pro:**

#-------------------------------------------------

#

# Project created by QtCreator 2017-02-17T11:27:36

#

#-------------------------------------------------

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

TARGET = QT\_Test\_Project

TEMPLATE = app

# The following define makes your compiler emit warnings if you use

# any feature of Qt which as been marked as deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += main.cpp\

mainwindow.cpp \

course.cpp

HEADERS += mainwindow.h \

course.h

FORMS += mainwindow.ui