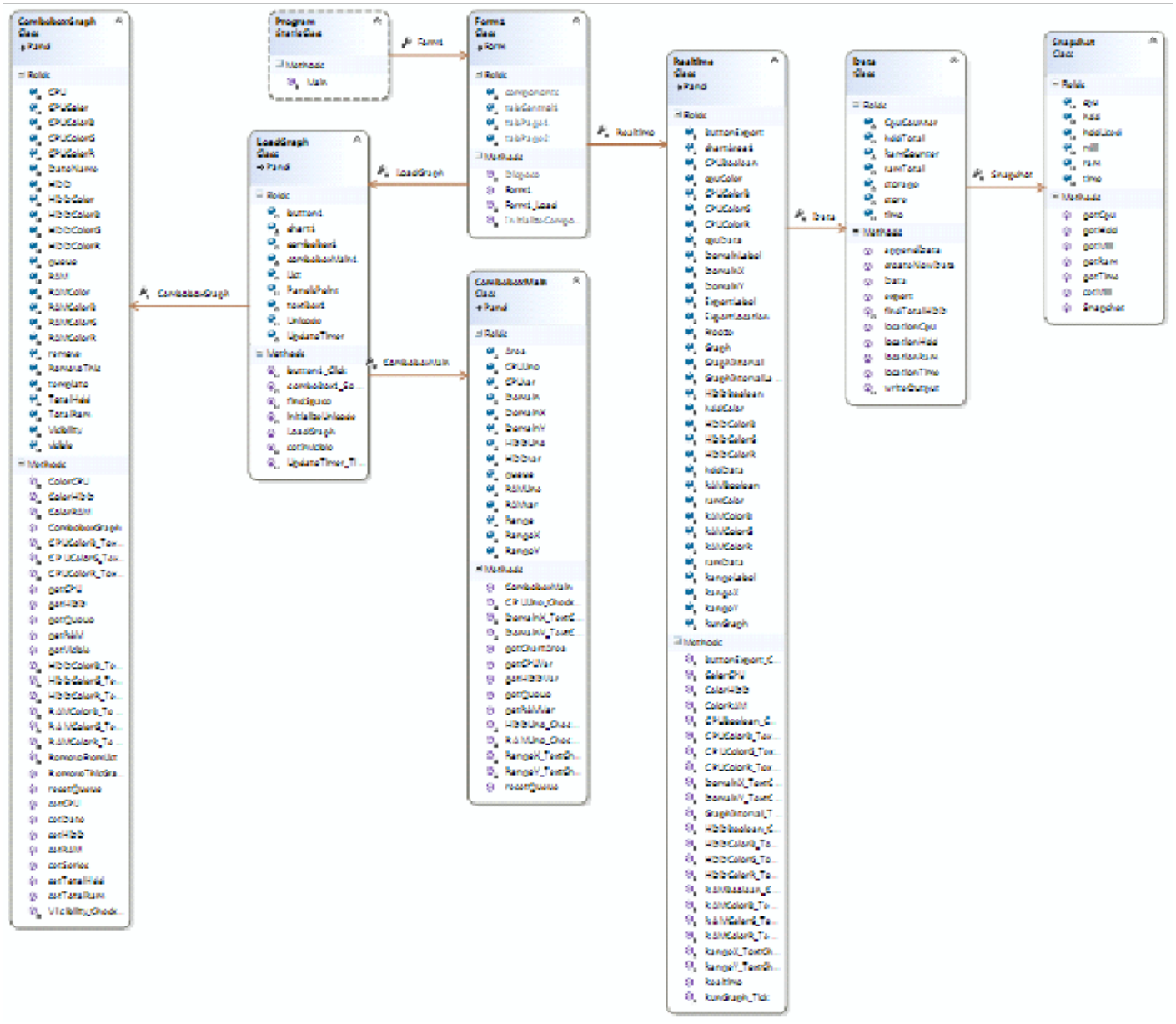
Criterion C: Development

Techniques used:

- Object Oriented Programming
- Recursion and dynamic data-structures
- Graphical user Interface
- Event based data acquisition and display management

Object Oriented Programming:

For the final program I have removed the DataManagement class, replacing it with the Data class. The Data class is renamed Snapshot. This program has 7classes, 4 of which are panels. I am using the in-built chart class in c# that draws lines for me and am just modifying its parameters to update lines. The diagram below was drawn with Windows Visual Studio 2012 Class Diagram.



By extending the panel component it allowed me to create my own isolated panel that could be marked as visible or invisible at ease without setting a large amount of code to repeatedly change the visibility of many graphics elements. The splitting up of the two graphs into different classes also allows me to more easily locate bugs in certain areas as well as modify them to adjust positions or add other elements.

Recursion and Dynamic Data Structures:

The use of LinkedLists in my program is a great success as often the program just needs to run through the entire list and the structuring of the linkedlist allows it to do this efficiently. I also do not know how long my program will need to run for and the dynamic nature of a LinkedList means that I don't need to worry about running out of space. The use of Booleans to check for changes also furthered the efficiency of the program when searching for a particular case. Unfortunately the inability to modify the Lines in just the classes means that an event based timer was required to search the list every 0.1ms in order to apply proper visibility of different lines as well as removal of a series from the combobox.

Fig.1 Code snippet of iteration to export data using a linkedlist

```
private void writeOutput(StreamWriter write)
{
    //iterates through list
    foreach (Snapshot piece in storage)
    {
        write.WriteLine(((double)piece.getMilli() / 1000) + " " + piece.getCpu() + " " + piece.getRam() + " " + piece.getHdd());
    }
    //clears list
    storage.Clear();
```

Fig.2 Iteration of LinkedList to search for Variables and apply visibility

```
private void UpdateTimer_Tick(object sender, EventArgs e)
{//use foreach in order to gothrough list and find value, can do the same for show visibility
    foreach (ComboboxGraph graph in List)
    {
        if (graph.getQueue())
        {
            if (graph.RemoveThisGraph())
            {
                comboBox1.Items.Remove(comboBox1.SelectedItem);
                chart1.Series.Remove(graph.getCPU());
                chart1.Series.Remove(graph.getRAM());
                chart1.Series.Remove(graph.getHDD());
                List.Remove(graph);
                setInvisible();
                comboboxMain1.Visible = true;
                break;
            }
            if (graph.getVisible())
            {
                if (comboboxMain1.getCPUVar())
                {
                    graph.getCPU().Enabled = true;
                }
                if (comboboxMain1.getRAMVar())
                {
                    graph.getRAM().Enabled = true;
                }
                if (comboboxMain1.getHDDVar())
                {
                    graph.getHDD().Enabled = true;
                }
            }
            else { graph.getCPU().Enabled = false; graph.getRAM().Enabled = false; graph.getHDD().Enabled = false; }
        }
        if (comboboxMain1.getQueue())
        {
            if (graph.getVisible())
            {
                if (comboboxMain1.getCPUVar()) { graph.getCPU().Enabled = true; } else { graph.getCPU().Enabled = false; }
                if (comboboxMain1.getRAMVar()) { graph.getRAM().Enabled = true; } else { graph.getRAM().Enabled = false; }
                if (comboboxMain1.getHDDVar()) { graph.getHDD().Enabled = true; } else { graph.getHDD().Enabled = false; }
            }
        }
        graph.resetQueue();
    }
    comboboxMain1.resetQueue();

}
```

Fig.3 recursion implemented to find certain Unicode characters

```
private String findSpace(StreamReader read, String hold)
{
    if (read.Peek() == 32 || read.Peek() == 13 || read.Peek() == 10)
    {
        read.Read();
        return hold;
    }
    else
    {
        hold += Unicode[read.Read()];

        return findSpace(read, hold);
    }
}
```

The piece of code above was necessary as the StreamReader of C# reads in Unicode and I created a dictionary in order to properly convert and check my document when I read it character by character

Graphical User Interface:

The Studio tool given to me was quite a shock and a lot of my time was learning how to manage graphics through exploration of the properties window and online sources. Working with panels, buttons and their chart for graphing. Once I got the hang of it the ability to drag and drop Graphics components greatly sped up the development of the Graphics user interface. It also allowed for the ease of adjusting the different elements in the case that I disliked a certain portions position or need to adjust due to size constraints.

Fig.1 Final GUI, Realtime graphing display

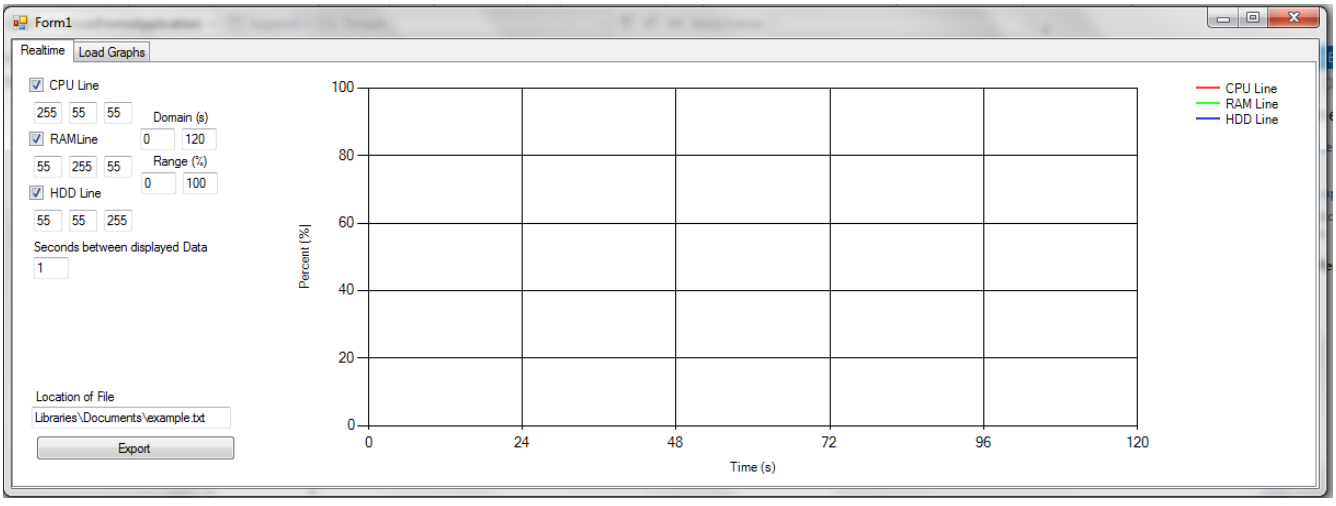


Fig.2 Final GUI, Load Graphs display

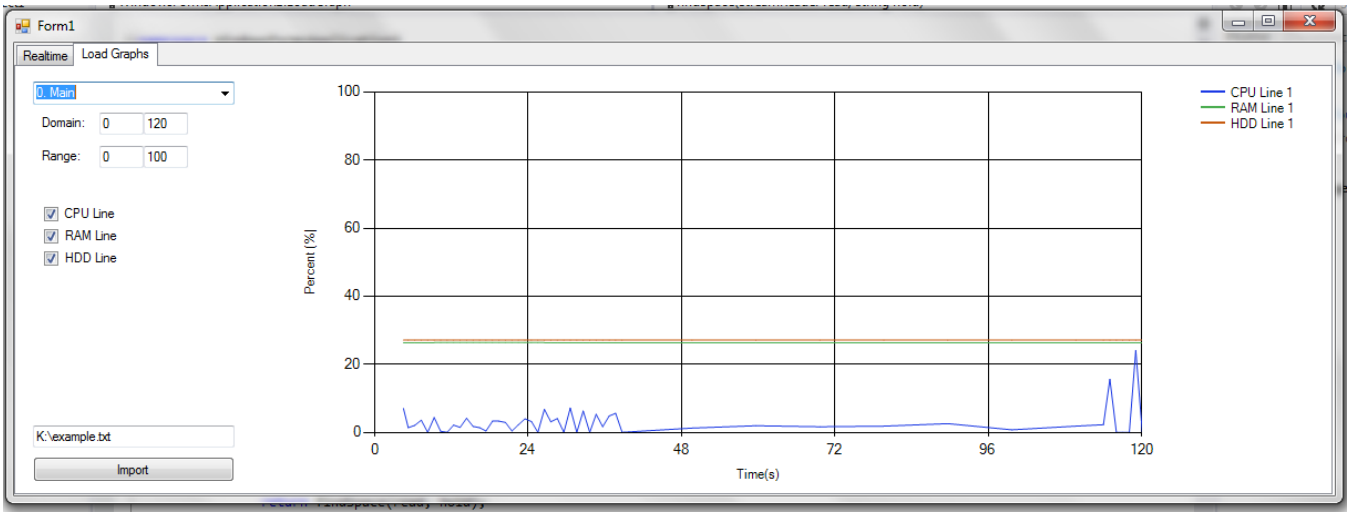
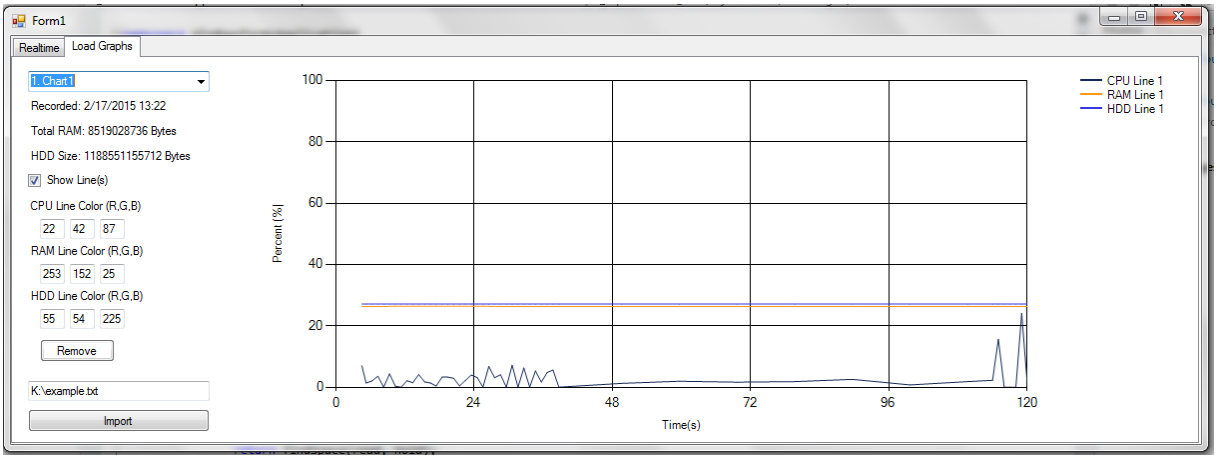


Fig.3 GUI of Load Graph display with user selecting a specific graph



Event Based Data Acquisition and display management:

In my java prototype I could run an infinite loop in order to keep the program running. This was because java managed the separate thread for me. However in C# I could not run such a loop without the Form starting, as such I used a timer. The timer created an event at certain intervals which allowed the graph to create data and process the data to display. I also used this timer event to check for changes to the LoadGraph panel as well. Due to the speed of c# the constant running of the timer in the background did not leave much of a trace. Other events include the regular button and checkbox events to change the various graph display settings and switch between realtime and loadgraph panels.

Fig.1 Code snippet of Data acquisition and display to graph

```
Data Freeze = new Data();

//does data display
private void RunGraph_Tick(object sender, EventArgs e)
{
    Freeze.createNewData();
    //tells data to create new info

    double time = (double)Freeze.locationTime() / 1000;
    //divides by 1000 since time is still in ms

    cpuData.Points.AddXY(time, Freeze.locationCpu());
    ramData.Points.AddXY(time, Freeze.locationRam());
    hddData.Points.AddXY(time, Freeze.locationHdd());
    //adds points to line

    Freeze.appendData();
    //adds points to file
}
```

Word Count 687