INITIAL REPORT, SENG3011 - SOFTWARE ENGINEERING WORKSHOP 3

Authors: Saishav Agarwal, Yuwei Chen, Rashim Rahman, Hariharen Veerakumar

---

1. Describe how you intend to develop the module and provide the ability to run it in standalone mode

The module is to be created such that it can analyse and, using the Momentum strategy, make choices to buy and sell stocks according to the data provided. The strategy will vary based on personal preference and, as such, we also require parameters. Finally, the module should ultimately be able to produce a result to a certain file. Therefore, to accompany the developed module, we will also require:

- the required file to be read in, in the correct format, that holds the input data
- A parameters file, that will hold the parameters required as per the user's choice
- The path of a directory to output the resulting file and log file to.

The application will be readily available for download, packaged with a sample file as well as the parameters file. These two input files will contain comma-separated value (CSV) data to be read in to complete the application input. The files are to be extracted into a specific location. Once this is done, the application will be run with the input file and the parameters file given as parameters for the application. The application will use the data read in from the files it is directed to in order to commit to the financial strategy the application is scripted to do.

The output data will be stored in another CSV file, since this holds the original format of the input. It is also universally used, and easily read in with applications such as Microsoft Excel. The log file produced will be in the form of a text (.txt) file. Again, this is readily legible by most text editing applications. Both these kinds of files are to be written out from the application once the results are produced.

Hence, provided the application within the module is provided with the input files (and their respective paths) in the module, and a path to create output to, the module will be able to function as a standalone application. Further refinement will determine what operating systems specifically will be supported. We will be aiming to cater for Windows, Linux and Apple OSs. (For further details on producing the standalone executable application, please see Section 4).

2. Discuss your current thinking about how a third party software system can invoke your module and interact with it.

As a standalone application, any third party software system can download the application via the download link on our webpage. The application will be packaged in a zip file, and can be extracted to obtain 3 individual files which are as following:

- A CSV file which is essentially the main input
- A CSV parameters file
- The executable application file itself

The most feasible decision involves development via Ruby, as two versions – one which is simply a module invoking the Ruby script, and the other module invoking the Windows executable (for Windows and, with Wine, for Linux).

The application can be invoked in different ways, depending on which operating system is used. In Windows OS, we can type the following command on the command line:

```
start [application-name].exe
```

If using Mac OS X, the Ruby script can be run with a simple double-click, hence acting as a standalone application from the module. The API/code will not be able to be modified or viewed in text. How ever, the script can be run as:

```
./[application-name].exe
```

For Linux, the module can be run with the same command as for Mac OS X, since they both use the 'bash' shell for their terminal. This can only be done if Ruby is already installed on the client syste m.

However, if they do not, they will need to run the application through Wine. By command-line, the following command must be run:

```
wine [application-name].exe
```

In either case, this will process the input and generate the required output file and a log file , as shown in the diagram previously. A default input file from the package will be run with the application. This will also be the course of action if the sample application is double -clicked.

Eventually, we expect our module to be available to third party applications more efficiently. With a GUI, there will be more flexibility in terms of modifying the parameters file as per the user's requirements. Also, the user will be able to select an input file of his/her choice. Finally, using these files as input the GUI will allow the user to invoke the module to process the file and generate the output files. The output CSV and log files will be stored in the user's computer. All of this will be done via an interactive graphical interface, which third party software systems can access using our webpage publicly.

3. Diagrams showing the overall software architecture

Architectures for Sections 1 and 2 are described with Figures 1 and 2 respectively.

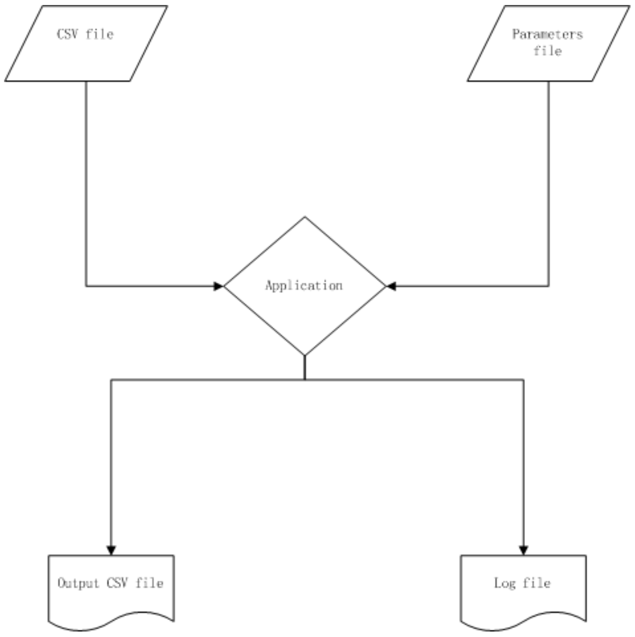

Figure 1: Visual description of how the application interacts with its inputs and outputs

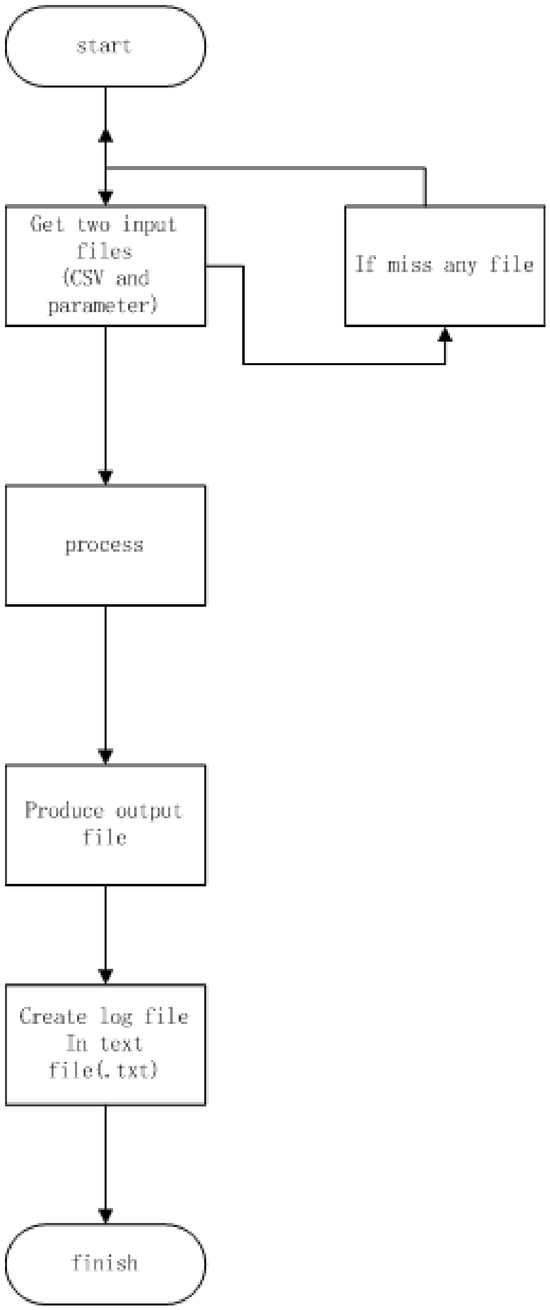
start
Get two input files (CSV and parameter)
If miss any file
process
Produce output file
Create log file In text file(.txt)
finish

Figure 2: the lifecycle of the application's usage from start of use to success

4. Implementation language and environment to be used

The module is to be written in Ruby. We have chosen this language for a number of reasons, above simply being most comfortable with the language.

Firstly, the parsing of information is made much easier using Ruby, making it a much more practical language for what we need. It is easily achieved using the regular expression tools that Ruby provides. Secondly, it uses principles of Object Oriented Programming – one which the developers of the module are strongly familiar with. These advantages are all encompassed in another functionality that Ruby provides. The files provided for input are all of a comma-separated value format. Ruby furthers the ease of using these files by providing classes to read in CSV files. Each record is then stored as an object, with characteristics stored as per its description within the CSV file, further ensuring a simplified program to achieve the aim.

Lastly, it provides a couple feasible methods to develop and execute our module – which are applicable for other languages, of course. However, in our considerations, we also considered conversion of scripts to actual usable applications. However, we must first consider which operating system we should use.

If we were to use the Windows Operating System, the available methods for running our module would be:

- Use the command prompt - fairly simple to use and effective, can run the module on the command prompt and direct input and output files to the module
- Use an IDE - very easy to use, can edit code while running, can easily monitor input and output files.
- Creating an executable ('.exe') file using a gem such as "Ocra"

We can choose to use either EditRocket or RubyMine IDE, which would make editing and testing the module very simple and effective. We could also visually control the input and output through the IDE. Both IDEs are compatible with multiple operating systems, meaning we could operate the module on both.

For options 1-2 however, one setback with using the Windows OS is that Ruby will need to be installed before use. Option 3, however, is a feasible option that does not have this setback, and has the full capacity to run as a standalone module.

If we were to use Mac OS X operating system, the available methods for setting up an environment, as well as running our module would be:

- The Mac terminal - fairly simple to use and effective, can run the module on the terminal and direct input and output files to the module
- Use an IDE - as mentioned above

The Mac OS X has Ruby pre-installed, therefore less processes to get the module running.

If we plan to use just the command prompt/ terminal we would need to run the module from it, and direct out input and output to it manually in our command. This is simple although during testing would be hard to make quick changes. The Mac OS X terminal would be the better option as Ruby is pre-installed, making it easier to get the environment set up.

The decision follows that, using the command prompt/terminal wouldn't require the installation of third-party IDE's, and hence would require less training for the developers. However, in using an IDE, which requires an initial installation, we will allow us to handle editing, input/output files and running the module in one window. Our final decision is as per Section 2, whereby there will be two different types of modules to account for Mac OSX, as well as Windows and Linux, separately.

5. Project Management Software

One of the major problems that we have encountered with software development in the past was collaborating as a team. To overcome this, we have decided to take project management more seriously.

We are using Pivotal Tracker for managing tasks assigned to each member in the team. Each task will consist of a description and a deadline. We have decided to create stories for each of the features that we expect our application to have. Tasks will be not only for development, but for testing as well.

We are also using Git for version control of the module. This helps us distribute the module easily between each developer, whilst allowing all changes to be made in one central location. Despite its complex nature, all users are proficient in how to use it and can use it in a powerful manner.

Apart from tasks, any bugs in the application reported by members of the team will be added as well. This will help us track our versions, and hence allow a better version control. Thus, Pivotal Tracker and Git complement each other well for our purposes.

With every release, we will include details of the release on the page too. This will enable us to get a clear picture of our current progress and backlog. Using this tool will help us communicate better, moni tor each other's performance and hence work more efficiently.