THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF ARTS

Jonathan Fenn
2019
THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
Bachelor of Arts

By Jonathan Fenn
2019

Development of the Data Extraction Utility SweetData

Author: Jonathan Fenn, Department of Mathematics and Computer Science, Author

Approved: Barry Myers, Ph.D, Department of Mathematics and Computer Science
Faculty Advisor.

Approved: Trent Holcomb, Process Engineer Amalgamated Sugar, Second Reader

Approved: Barry Myers, Ph.D, Chair Department of Mathematics and Computer Science
Development of the Data Extraction Utility SweetData.
FENN, JON (Department of Mathematics and Computer Science), MYERS, DR. BARRY (Department of Mathematics and Computer Science).

SweetData is a utility to extract data and convert it to information reports. It is to replace an outdated utility used by Amalgamated Sugar. It uses a GUI (graphical user interface) instead of a console interface and is written in C#. Designing software is a process that can be difficult, no matter how simple the job. It takes planning to be done efficiently and what the developer envisions is often different than what the end-user envisions. Using Agile development helps to alleviate that problem. Programming languages are similar and learning new languages is easier after learning your first. A starting point for designing an application is the GUI. The GUI helps to visualize the framework of the application. The GUI should allow the user to navigate the application without confusion. Binary files had the data that needed to be extracted and processed into information. The information is exported to a CSV (comma separated values), text, or Excel format. The reports show date ranges with current or YTD (year to date) values. The use of multithreading allowed the report to be written while doing other stuff. This lowered the runtime from seconds to milliseconds. Future versions will have PDF support.
Table of Contents
Title ........................................................................................................................................... i
Signatures ................................................................................................................................... ii
Abstract ....................................................................................................................................... iii
Table of Contents ........................................................................................................................ iv
Acknowledgements ...................................................................................................................... v
Development ................................................................................................................................. 1
  Requirements ............................................................................................................................... 1
  Proposal ....................................................................................................................................... 2
  Planning ....................................................................................................................................... 4
  Graphical User Interface (GUI) ..................................................................................................... 7
  Accessing Data ............................................................................................................................. 10
  Exporting Information .................................................................................................................. 10
  Deliver to Customer ..................................................................................................................... 11
Problems ......................................................................................................................................... 11
Solutions .......................................................................................................................................... 12
  Agile ............................................................................................................................................ 12
  Multithreading ............................................................................................................................ 13
Conclusion ......................................................................................................................................... 14
References .......................................................................................................................................... 15
Appendix .......................................................................................................................................... 16
User Manual ..................................................................................................................................... 16

Figures
GetQ (Figure 1) ............................................................................................................................ 2
Proposal (Figure 2) ......................................................................................................................... 4
Ghantt Chart (Figure 3) .................................................................................................................. 6
Revised Ghantt(Figure 4) ................................................................................................................ 7
Configuration (Figure 5) ................................................................................................................ 8
GUI (Figure 6) ............................................................................................................................... 9
Multithreading (Figure 7) .............................................................................................................. 13
Acknowledgements

I would like to thank my family Jammie, Josh, Bentley, and Anna for supporting me and for the sacrifice of work, school, and family. I would also like to thank my co-workers for working with me to allow me to attend class. Thanks to my Professors at Northwest Nazarene University for pushing me and encouraging me.
Development of the Data Extraction Utility SweetData

Amalgamated Sugar is the second largest sugar producer in the United States and making sugar is a complicated process. There are many variables and a lot of paperwork to perfect the process. SweetData is a utility to extract data and turn it into information for use by the Chemists at Amalgamated Sugar. It was made to replace an outdated utility written in PASCAL. The old utility had a console-based UI (User Interface), and SweetData uses a GUI (Graphical User Interface). SweetData is more user-friendly than the old utility. Software development is a process and if not done right can cause unnecessary work.

Development

The first step in developing software is to get a project. That sounds easy, but often it can be one of the hardest parts of the process. Jeff Durning, Chief Chemist at Amalgamated Sugar, wanted to replace an old utility with one that was easier to use. He recruited a software developer to develop it.

Requirements

There must be a reason for software to exist and it is the job of a software developer to help figure out what the customer wants or needs. The requirements need to be clear and to the point. The developer should have most of what he needs after this stage of development.

Jeff explained that the old utility read data from a PDF file but used a console line interface that was confusing as shown in figure 1.
In the ledger box, the user would type what page and row the information that they wanted is located. The ledger file is a text file with a list of all the tag descriptions. All the data is sorted by a run type. For example, Beet run, would hold all the data for the time Amalgamated Sugar is cutting beets. Each run’s data is in a different file. The new utility would have to pull from a PDF file to extract the data needed.

The old utility was not very user-friendly. The user had to know the page and row number and you had to enter where the ledger file was every time. Jeff wanted it easier to use. For it to be useful to him, it needed to process the data into reports. He wanted PDF, text file, CSV (Comma-separated value), and Excel reports. The developer felt comfortable to accept the project and that he had enough information to get started.

Proposal

Every organization will have an approval process in determining if the software meets their needs (Tilley). Amalgamated Sugar and NNU (Northwest Nazarene
University) is no different. Dr. Myers needed to approve that the project would meet the requirements for a senior project. Ian Kiharra, the Corporate Chemist at Amalgamated Sugar, and Amalgamated Sugar’s legal department needed to approve it to make sure it did not release any information detrimental to the company.

Dr. Myers received the first proposal. Figure 2 is that proposal. He approved the project. It met the requirements, and it gave real-world experience for developing software. It was stated what the program needed to do, how, and why. The cost was not an issue, but in most projects, it would be. The triangle to see if a program is worth doing is Time – Cost – Scope (Tilley). There was no cost so Time and Scope could be raised.

The same proposal went to Ian Kiharra. The first problem of the project happened at this point. The data was not stored in PDF files but daily binary files. The proposal was revised and resubmitted to Ian.
Figure 2

The final step in the approval process was submitting the revised proposal to Amalgamated Sugar’s legal department. Companies do not want proprietary information to get out to their competitors, and there are legal concerns with dealing with information. A company must protect itself. The legal department approved the proposal contingent on a signed NDA. (non-disclosure agreement)

Planning

The fun part of developing software is coding it. There is much more to it though and planning is crucial. Most times if the developer does not plan more work is created,
the code will probably be poorly written and will be prone to errors. The software
developer needs both a plan on how to solve the problem and a schedule to do so.

The developer needs to decide what language they are going to use to write the
software. This decision helps you to make several important decisions. What libraries are
out there, what IDEs (Integrated Development Environments) are available, and does the
language have the capability to do what was needed?

C# was chosen because there are some libraries to help with PDF files and Excel
files. It was a new language and it was an opportunity to learn a new language. A
language that is widely used at this moment. Tutorial’s Point had a good C# tutorial
(Tutorialspoint).

Visual Studio was the IDE that was used to write the utility. It makes things easier
if a developer is already comfortable with the IDE and does not have to learn to navigate
a new IDE. Microsoft Office Interop is a resource that Visual Studio already has. Using
something that is already there makes things a bit easier.

A name for the utility is needed. It seems like a trivial thing, but it is important.
The name will be what people remember the application as. It needs to be something
professional and inventive at the same time. The first name, “Report Generator,” is too
boring. It does explain what the program does, but it is not something that has flair. The
second name that was considered, “SKP,” is a fictional country in an RPG game created
by Paul Tesch Jr. It had flair, but it does not go with this utility. It also is a name from
another person’s work. “SweetData” was the name chosen. It has flair and says
something about what the utility will do.
When planning a schedule, it is important to be as specific as possible. A developer should break down each task as small as possible. If one of the tasks takes a large amount of time, it needs to be split up into smaller tasks. When the schedule for SweetData was made, it was too general and had to be split down further into different tasks. The final schedule that was approved is figure 3.

The first task “Learn C#,” could have been split into more general tasks, but the idea was that learning the language would be an ongoing process. The project would continue as the language was learned. Each topic was one to two weeks and only covered one aspect of the process. It continued throughout the entire process from the start until the utility was finished.

Even with the best-planned schedule things happen, this utility was no different. Due to circumstances, the project was delayed a semester. It meant an updated schedule
and figure 4 is that schedule. Schedule changes can happen a lot. Software developers come across unforeseen circumstances and must update their schedule. Possible instances are lost or delayed funding, the customer wants something new, or the company wants to release the software quicker.

![Gantt Chart](image_url)

**Gantt Chart**

**Graphical User Interface (GUI)**

The GUI is the skeleton of the application and is the best place to start. The design allows the developer to work on each aspect of the utility and allows the developer to make sure each aspect works.

The utility relies on knowing where to find the data it needs to use. The user needs to specify where those are found. SweetData checks to see if there is a configuration file. If there is not, the configuration screen is visible. The design is shown in Figure 5. Amalgamated starts each year on the day we start the Beet Run. That is either the start or close to the date that we start slicing beets. There is a Datebox to set
the start of the Beet year. It currently does not have much use, but a future version would keep track of the days that do not count and be able to date the reports by calendar dates instead of Operating days.

Amalgamated Sugar has four plants, and if SweetData was going to be useful at the other plants, it would need to be able to find their data as well. It was decided that the user would be able to determine where the data files are kept by setting an absolute pathname, and SweetData would use a relative path formula to find all the files. All four factories store the daily files in a different network directory, but they all have the same organization in that directory.

![Configure](image.png)

**Figure 5**

Figure 6 shows the main GUI for the first version. It has a few buttons, two list boxes, two sets of radio buttons, and two number boxes.

The two number boxes are used to set the operating day ranges. Possible values are one to three hundred sixty-five. The lower range cannot be higher than the upper
range. The utility will update the upper range to equal the lower range if this happens.

When the number changes the utility makes sure that nothing is out of range.

![Image of a software interface with list boxes and radio buttons]

**Figure 6**

The two list boxes are for the unselected tags and selected tags. The tags are found in a ledger file that had already been written. The work had already been done, and it was easier for the program to read in the data, rather than hardcode it in. It also allows for changes to be done without having to recompile the program.

One of the groups of radio buttons was to select the Run the user wanted. Throughout the year different processes are going on. Beet Run is when the factory is slicing beets, Juice Run is making sugar from the juice, and Separator Run is further separating sugar from molasses to get as much sugar as possible. When the user selects a run, the utility reads in the tags from the ledger file and places any not already selected into the unselected box. As the user select items, they are moved over to the selected box.
The second group of radio buttons is the format the user wants to export. It is required that there be at least one item selected before a report can be generated and exported. SweetData reads in the data for each day and formats it into the report the user wanted.

**Accessing Data**

To read the data, SweetData needed to know the location of the files. Jeff did not know where these files were stored. The developer had to explore the network and find out where the files were stored.

Binary files are tricky and to be able to retrieve the data from the files the utility had to know what data types were stored. The developer was able to obtain the goad from the old utility and see what data types were used.

**Exporting Information**

SweetData iterates through each desired operating day. It loads the data from the file into an array. The utility prepares the report as each day is read and processed. The report is saved with the desired filename and format.

There are libraries to manipulate PDF files, but they are still a bit complicated and some have a fee. ISharp was the library that was going to be used, but Jeff decided that PDF support was not necessary. The decision was made to drop PDF support.

The text file and CSV formats were easy to implement. The text was formatted to output the data into a text file. The CSV has all the information separated by commas. The Excel report needed a little more formatting than the text report and CSV report. Microsoft Interop made it possible easy to do so (Csharp n.d.b). It was easy to autofit the columns and even color them.
Deliver to Customer

Delivering the product to the end user can be a bit nerve-wracking. The customer may have had a different vision and they may not be happy with final product.

SweetData was delivered to Jeff. The developer installed it, gave him the user manual and did a quick demo for him. While he was playing with it, he caused it to crash and found some other bugs that the developer had not found. Jeff tried to explain to the programmer what was happening, but Jeff did not how to explain it well enough for the programmer to figure out what was happening. Jeff did not like how the excel report was formatted. It was not easy to do the calculations he wanted in Excel.

Problems

There were a few problems that came up during the Development process. Almost no software development will go as planned and overcoming those issues is an important part of being successful.

The first problem was the source of the data. Jeff thought that the data was pulled from a PDF file. He thought this because in the old utility he had to put in a page number and row. During the time the developer was searching through the network, he found ledger files for each run. The page and row was the location of the tag in the ledger file. The developer used this ledger file to load in the tags and used the same format for the array. This problem was found when the proposal was sent to the Corporate Chemist. The proposal was revised and resubmitted to the Chemist. This shows two problems a developer could encounter. The customer did not know what he wanted, and proposals may have to be revised.
The next problem was the format of the binary files. The developer did not know how the data was stored in each daily binary file. Fortunately, the developer was able to get a copy of the old code and decipher the data types.

Another issue that came up was that the software developer had to take some time off. Life is not perfect, and things can happen that will delay or cancel a project. The delay caused the work to get behind in schedule, and a new schedule had to made. The new schedule is Figure 4.

Jeff was the customer for the project and the utility was not what he needed. It did not do everything he wanted. It is the job of Software Developers to help the customer tell us what they want or need.

The Excel report was too slow. Users are impatient, and if they do not see things happening right away, they think it is broken.

**Solutions**

**Agile**

Some of the problems could have been alleviated by using Agile Methods. By doing a small part and bringing it to the customer, the developer can make doing a small part and bringing it to the customer, the developer can make sure that the work they are doing is what the customer wants. Also, if the customer decides they want something else, will be easier to change while it is being coded. (Tilley 11.6)
Multithreading

One of the teams in Fall 2018 Operating Systems class added parallel processing to SweetData. They wanted to see how much faster it would run. They used a task to run the Excel export simultaneously with the main thread.

```
Task.Factory.StartNew(() => createExcel(thelist));
```

This code creates a new task and runs `createExcel(thelist)`. They used `StartNew` so that a parameter could be sent. If `Task.Factory.Run(() => createExcel);` was used a parameter could not have been used. There would have been a little less overhead, but they felt the slight overhead was worth it. C#'s stopwatch class was used to time the execution from pressing export until the selected list cleared (Microsoft). The results are shown in Figure 8.
An issue with the stopwatch class is that it is only accurate to 1/32 second or 31.25 milliseconds. In the worst-case scenario, it took 9.3 seconds to go from export until clearing the selected screen. With parallel processing, it lowered the time down to 23 milliseconds. The most that can be said is that it is less than 31.25 milliseconds. It is still is an amazing result.

**Conclusion**

By designing this utility, I learned several lessons. It takes planning to output quality software, you must look outside the box and try to learn new things, languages are very similar, and it is easier to learn more languages after you learn the first. You must understand what the customer wants and needs. It is easier to fix things as you go, rather than after you have finished everything.
References


Tilley, S., & Rosenblatt, H. *Systems analysis and design* (11th ed.).


Configuring SweetData

SweetData must be configured before it is fully functional. It will check to see if it is configured and if it is not the configuration dialog will pop up

- Setting Beet Run Date -
  In this version it is not necessary to set the date.

Setting Path Name
Set the path to where the Beet and Sep directories are located

- Click the okay to save any changes and cancel to ignore any changes

About

About will show information about SweetData including the version number

Selecting Run

- Select the run where the desired data is found
- A run must be selected to select data
- If SweetData is not configured an error will occur prompting, you to configure it

Unselected Data

- If the run has been selected a list of data points will appear
- Click on the data point you wish and press the right arrow
- Data points from different runs may be selected

Selected Data

- To unselect data, select the unwanted data point and press the left arrow

Operating Day Range

- Select your starting day (1-365)
- Select your ending day (1-365) Must be higher than the starting day
Exporting Data

- YTD - Checking this box will show the Year-to-date values
- Select Report Type
  - CSV – Comma Separated Values
  - Excel
  - Text
- Press Export – Name the export file