ProductivityPlus Technical Report

A technical look at ProductivityPlus, a Java-based GUI Windows applications used to collect and view data on how much time is spent in various applications.

Austin Ayers

C S4800-101 CAPSTONE PROJECT

Prof. Frank Barry

Appalachian State University

Abstract

ProductivityPlus is a simplistic Windows GUI application designed to record and analyze how much time the user spends in the applications they use. All the saved data can be viewed in-application with charts and graphs or exported locally to a .xlms file extension to be viewed in Excel or any other spreadsheet application that opens .xlms files. The program works by using the Windows API to check what program a user is in and saves the data locally after each program swap. In ProductivityPlus' current state, users can track everything they do, and filter the results to display the set of programs they have data about, which also applies to old saved data. All the user's data stays on the local machine, so the user can rest easy that their collected data is never being moved around or seen by anyone but themselves. ProductivityPlus gives the user a one-stop-shop to collect, view, analyze, and export data that they collect on themselves.

Keywords: tracking software, time management, time analysis

Contents

Tables

Figures

Introduction and Project Overview

The foundation of ProductivityPlus is its main function; the program timer. The point of the ProductivityPlus project was to create an easy to use program that allows the user to track how much time they spend in the applications they use and give the user control to selectively view their data in-application with graphs and tables or export their data to an Excel file (.xlms). ProductivityPlus plays the role of a minimalistic program tracker, this is useful because finding a free program tracking application is challenging because the nature of the application itself is a double-edged sword. To explain, users are allowing tracking software onto their machines, something most people are usually very much against. Most program tracking software also has a private source, so viewing exactly what is going on with the information gathered and how much is being gathered is limited to what is made readily apparent. This created the desire to create an open-source public application that performs these tasks, where users can track what they do and feel safe doing it.

What should a program tracking application do? When developing an application that is going to collect data for the user to examine, there needed to be a way to export the data to view in external more robust applications such as Excel, and a way to view the information locally inside the application itself. These were two very important objectives for ProductivityPlus to deliver on.

Being able to select which programs to display and export was another important objective. The user needs full control over their data, without being able to quickly and easy narrow the collected results viewing data is more challenging. This problem was resolved by adding lists to remove items from the display table, or to only display a few items that are present inside of the inclusion. *Table 1* elaborates what this solution is solving.

Collected Data (what is stored to be filtered)	**Display Mode and List (the lists and their contents)**	**What is Displayed**
ProductivityPlus – 30 seconds *Google Chrome – 25 seconds*	Inclusions Filter: Google Chrome	Google Chrome – 25 seconds
ProductivityPlus – 30 seconds *Google Chrome – 25 seconds*	Exclusions Filter: Google Chrome	ProductivityPlus – 30 seconds

Table 1 – Display filter example

Allowing the user to load old data and set the same filters to manipulate how the old data was viewed was another important objective, which also lead to the desire to be able to look up information over a date range. With the given filters inside *Table 1*, any program can be hand selected to be viewed from the current day's reading or any number of days from past saved data. Allowing old and current data to be viewed in-application with graphs was another useful objective, especially for those without access to a third-party data analysis tool. Finally, giving the user various settings that save and load between launches of the program was very important because of the different ways to filter data. Forcing the user to have to reset each setting for every launch is not user friendly is makes the application more challenging to use and less user-friendly.

Potential Users

The users of ProductivityPlus is potentially unlimited. Anyone can own a computer, and anyone can develop the desire to track what they do on their computer. Clearly not everyone will, but with this idea in mind it was important to develop the program around the possibility of a large range of different types of users. There are many different reasons someone would want to use ProductivityPlus, varying from users trying to see where they waste their time most or simply those who want to keep records of what programs they use and when. Besides the type of user that simply wants to keep track of what they are doing, developers and other freelancers can keep time logs of how long they spent in various projects and manipulate their data further from that point. Since the data is so easy to view and access, this widened the audience to people who simply want to keep a history of what they do and when they have done it.

Existing Works

Other work does exist such as RescueTime (RescueTime.com) and Toggl (Toggl.com) that offer similar services. Both applications differ from each other and ProductivityPlus, along with what the end goal is for the program. RescueTime seemed to only selectively track which programs it collected times for. This could be something to do with the delay between the local application and the fact that they force you to view your data through their dashboard. Toggl differs from ProductivityPlus, but both share similar ideas and features. Toggl is more group-work oriented and more relatable to businesses and getting projects completed. This takes what ProductivityPlus does to a different direction and does not fulfil what the main objectives set for ProductivityPlus. One major difference is that both Toggl and RescueTime force their users to interact with their application and service through a website dashboard.

Value in ProductivityPlus

ProductivityPlus was designed to maintain simplicity because of the possibility of a wide array of users and user backgrounds. This is accomplished by arranging many of the interface objects into the main window and placing the less used interface objects inside of easy to find windows that can be accessed from the main GUI window. Alongside having an easy to navigate GUI, there is also a link to an instructional video on how to use the program embedded in the About window following the description of each feature. Users have access to most of the features in one location and at the push of a button. Everything that is collected never leaves the user's machine, same for the results that are created in-application. The results can be used any many different third-party software since Excel files are portable and common.

Table of features

Feature	**Description**
Program Timer	Start and stop the timer to track what you are doing and for how long.
Graphical Output	View the top results from a collected set of data in a pie chart and bar graph.
Export to Excel	Export collected data from a single date or range to a .xlms file.
Single Program Search	Search for data about a single program over any time range.
Consolidate Programs	Combine sub-entries from a parent program into one entry with a combined time.
Load Old Data	Load a single data or date range of old saved data into the program.
What to Display	Selectively display or export data entries.
Custom Time Units	Time units can be changed between seconds, minutes, or hours.
Update Notification	Alert the user when an update is available.

Table 2 – Program features

Design, Development and Test

Design

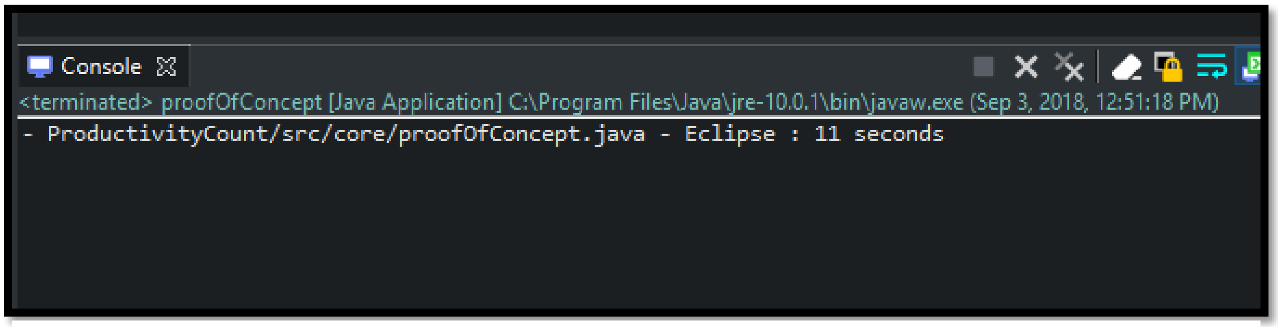
ProductivityPlus was separated into three packages, core, gui, and graphs. The graphs package contains the two classes the generate the bar graph and the pie chart using an open-source Java library framework called JFreeChart [9]. The gui package contains all the Swing [5] classes that build the various windows present in the application. The Main class can be found under the gui package, this is here because the class begins from the Program Timer window, which is present inside of the Main class. All the other features are present in the core package.

Each core feature has its own class that performs its duties. The Excel export option has a class that performs all the Excel export features, but this class and many others rely on classes such as the class that converts time and the class that handles the data. A good way to think of the core classes is that they take advantage of the data handling class and the time conversion class. The data handling class (DataHandling.java) handles all the saving, loading and HashMap (entries are stored in key-value pairs for program name and time) manipulation. Many other classes need to access saved data or filter a HashMap, so they would need to use a method inside of data handling, which is why it is so important. Every time a data entry is displayed, before it was displayed the time unit was cross referenced with the user's preference and likely converted using the time converted class (TimeConvert.java). The sole purpose of this class is to handle all of the time conversions, including pluralizing correctly, such as displaying the incorrect string of "1 minutes" as "1 minute."

To write to Excel files, the Apache POI [4] library is used to streamline the document writing process and to make the result an actual .xlms file. This library is very useful in the fact that it makes creating Excel documents extremely easy and efficient. MiG Layout [5] as my Swing layout of choice. The reason I chose to use this specific library layout is because of the great flexibility it gives the developer to specific pieces of a GUI. The MiG Layout [5] allows creation of as many row and columns as needed, allowing for more panels to be placed inside of created cells which can also apply the MiG Layout to, which would further increase flexibility of placement of GUI elements. To give the user easy access to a calendar and an easy input for dates, I used the LGoodDatePicker [6] library to create calendar elements that take input from the user.

Development

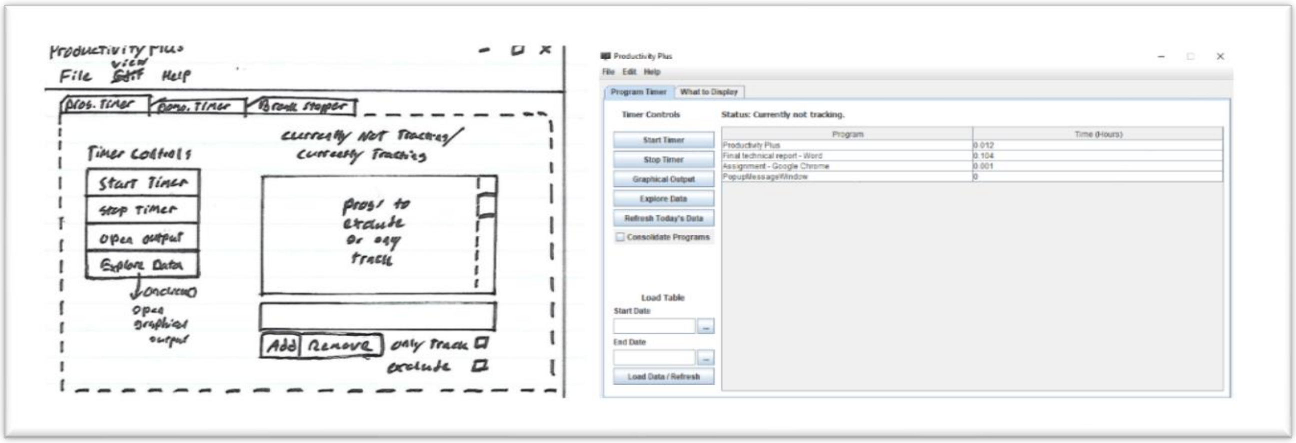
Moving on from designing the application and to developing it, experience with Java and building small GUI applications was useful, so the biggest risk area was being able to complete the core task of tracking the user. The first step of the project was to make sure that task could be accomplished. Learning the basics of using the Windows API with JNA [8] and writing a simple Java loop that printed to console how much time was spent inside of the development workspace showed that the project was doable, and the core feature was already almost completed.



```
<terminated> proofOfConcept [Java Application] C:\Program Files\Java\jre-10.0.1\bin\javaw.exe (Sep 3, 2018, 12:51:18 PM)
- ProductivityCount/src/core/proofOfConcept.java - Eclipse : 11 seconds
```

Figure 1 – Proof of concept

The proceeding steps of the program development were to draw out what each GUI window should look like. Here are the sketches that were used to plan out the main GUI compared to the actual result.



Status: Currently not tracking.

Program	Time (Hours)
Productivity Plus	0.012
Final technical report - Word	0.104
Assignment - Google Chrome	0.001
PopupMessageWindow	0

The windows were promptly created after their blueprints were completed to solidify groundwork for all the future features. All the important features were planned out and ordered by importance. Starting with the program timer first, moving onto saving and exporting, and displaying the information in the table. Once all the data was available and usable, the other features such as unit

Figure 2 – Gui blueprint comparison

changing, and graphing could be added. The data had to be easily accessible for most of the other features to have any work done or to be tested.

Throughout the development process, many changes took place, such as removing the Pomodoro Timer and Break Stopper features. I felt that I was trying to extend ProductivityPlus into areas it did not necessarily need to go, and I should focus on fine-tuning the program timer features and work on usability.

Testing

The Program Timer feature is currently the only feature with a jUnit test class. All the other features are proven by demonstration or by creating dummy data to view the results of various functions. To see if the graphs worked, simple program record data was created and chosen to be viewed with the graphs to see if the data was accurately displayed in both plots. To test exporting, the same approach was taken. More data entries were created and exported to be checked in Excel to see if it matched the data inside ProductivityPlus. Times were validated using the local system clock on the PC used for development. This process was repeated to check for loading and saving date ranges. The process of creating data to test features was used in every other core feature of the program. Due to the nature of the program allowing users to view all their data, it is easy to create data entries to test to see if features work.

Results

I am proud to say that ProductivityPlus has met the majority of the project's objectives. If a feature was unimplemented, it is because it was scrapped from the project due to the idea of the project changing directions or the feature becoming obsolete. Every core planned feature relating to the Program Timer has been implemented and works. *Table 3* shows the state of each core feature.
The application creates an easy to use interface to most users to be able to understand how to track what they do. The program collects data from everything the user is doing, and then filters through the entire collection of data to display the results according to user settings. The user has many ways to customize what the program does and how it looks, such as customizable units, consolidation, and graphing. Allowing the user to use the data outside of the program is also available with Excel exports. The program timer passes the unit tests for randomized program launches and times; in other words, it accurately tracks information. The program is set to check what program the user is in once every second. This value was chosen to get as close to an accurate measure as possible without creating

unnecessary checks. There is still a chance the time is going to be off a bit, but the value would be less than a second. Scalable time checks can possibly be a future feature to improve the chance of a slightly incorrect time.

To monitor how much CPU is used when the program tracker is running, the management interface for the host machine's JVM, OperatingSystemMXBean [1] was used monitor CPU usages (Intel i5 3570K @ 4.2GHz was used for testing) by using System out to print the ratio from the getProcessCpuLoad [2] and getSystemCpuLoad [3] methods inside of OperatingSystemMXBean. Both methods return a value from 0.0 to 1.0, which can be converted into a percentage to obtain a better idea about how much CPU is used. To test how much CPU is used, the program tracker was executed for 30 seconds to obtain CPU data from both methods. All the data obtained (decimal values) were then averaged to obtain an average CPU usage value from both methods. The results conclude that when the program is running, 0.017957669 (1.7%) of the Java Virtual Machine process was being used by ProductivityPlus and 0.033469234 (3.3%, 86% increase) for the whole system. In comparison, when ProductivityPlus is idle and not tracking, 0.014649380 (1.4%) of the Java Virtual Machine process was being used by ProductivityPlus and 0.020728765 (2.0%, 41.5% increase) for the whole system.

Table of features met

Feature	**Current State**
Program Timer	Fully functional; unit test included.
Graphical Output	Fully functional; manual test
Export to Excel	Fully functional; manual test
Single Program Search	Fully functional; manual test
Consolidate Programs	Fully functional; manual test
Load Old Data	Fully functional; manual test
What to Display	Fully functional; manual test
Custom Time Units	Fully functional; manual test
Update Notification	Fully functional; manual test

Table 3 – Features met

Problems Encountered

During the development, many problems came up simply because of knowledge gaps about many of the features that needed to be implemented. An example being setting a tray icon for an application or using a Swing table. All these problems were relatively easy to resolve without much trouble. The biggest problem with this project was handling the actual data. Since all the data is serialized and stored in files, each time data is needed it must be reloaded into the program and read from the file. The challenging part is keeping up with the conversions after reading the file, especially if a date range is used. If a range is used, all the maps must be combined while keeping the consolidations and display modes in mind. The data must pass through several classes before being displayed. The larger the project became the more challenging it became to ensure all the previously added features still worked. Maintaining functionality while growing the program became challenging for me, but I resolved this by expanding functionality and separating methods into their own classes to keep methods and classes as close to following the single responsibility principle as possible.

Conclusions and Future Work

Wrap Up

Productivity was built so that it would be easy for the average computer user to use it, since the average computer user could find use in ProductivityPlus. After creating a list of useful features revolving around time tracking and productivity management, work began on the application itself. I learned much about time management from building ProductivityPlus. Time was one of, if not the most valuable resource all throughout this project. I also learned how important planning is when it comes to a project. The better the plans are, the easier the project is to program. If problems can be seen in advance, solutions can be created before the problem is at hand and hopefully resolve the problem faster than it would have been resolved without planning or forethought. Using Git to backup my code was another important lesson learned. Reverting to previous versions and trying out features that I was not sure was best for the project was easy to accomplish because of Git and the usage of branches. I learned that usability is an entire world by itself. It is very challenging to determine how the end user will use the software, so finding issues that users would find because they use the program differently than intended can be challenging. The project resulted in a fully functional desktop Windows GUI application that allows the user to complete the task of easily tracking what they do and for how long.

I also learned that adding comments as you code is very important, and tricks such as different naming conventions for different variable types is also useful. An example is using camel case naming for non-static variable and underscores for static variables, this way anyone reading can identify the different immediately without needing to trace through code.

Future Work

Since I use ProductivityPlus every day I will be maintaining and adding more features to it. There is always room for future work and improvement. One pressing concern is that not all features have unit tests, in fact most of the tests for the project are based on manual creation and manipulation of data to prove that features work. Writing unit tests for the more challenging parts of the program to write tests for would be a future improvement to the integrity of the application. Adding more graphs to display user data would also be a nice feature, especially for the users that do not wish to use third-party software to view their data. Finally, improving the program consolidation to be able to consolidate the children of a parent entry into subcategories would be a nice feature to implement. An example of this would be combining tabs of different websites inside of Google Chrome to the individual website, instead of mapping all Google Chrome entries to a single item.

References

[1] OperatingSystemMXBean. The management interface for the operating system on which the Java virtual machine is running.

[2] getProcessCpuLoad. Returns the "recent CPU usage" for the Java Virtual Machine process. This value is a double in the [0.0,1.0] interval. A value of 0.0 means that none of the CPUs were running threads from the JVM process during the recent period observed, while a value of 1.0 means that all CPUs were actively running threads from the JVM 100% of the time during the recent period being observed.

[3] getSystemCpuLoad. Returns the "recent CPU usage" for the whole system. This value is a double in the [0.0,1.0] interval. A value of 0.0 means that all CPUs were idle during the recent period observed, while a value of 1.0 means that all CPUs were actively running 100% of the time during the recent period being observed.

Libraries Used

[4] Apache POI. Read and write Microsoft Excel files using Java.

[5] MiGLayout15-swing. The Swing user interface layout used to constrain and place the objects in the user interface.

[6] LGoodDatePicker. Create calendar widgets in Java GUI applications to allow users to easily pick dates.

[7] jUnit. Unit test library to write tests for Java programs.

[8] JNA. Java Native Access; easy access to native shared libraries without using the Java Native Interface. This is used to access the Windows API to track in-focus programs.

[9] JFreeChart. Create Java graphs and charts with various data inputs.