SENIOR CAPSTONE DESIGN DOCUMENT

NOVEMBER 23, 2019

SOFTWARE INNOVATION FOR DUAL SCREEN NOTEBOOK

OREGON STATE UNIVERSITY
CS 461, FALL 2019

PREPARED FOR

INTEL

MIKE PREMI _____ Signature _____ Date

PREPARED BY

GROUP 66B

AUSTIN WILMOTH _____ Signature _____ Date

BAO PHUNG _____ Signature _____ Date

HARRY MILLER _____ Signature _____ Date

JUSTIN PARKS _____ Signature _____ Date

SONICA GUPTA _____ Signature _____ Date

Abstract

Our group will be building an application whose main function is to provide an easy setup for users who may use the notebook for a variety of purposes, such as music production, coding/productivity, and gaming/streaming. To provide tools to music producers, we will provide integration with a DAW, which requires programmable keyboard shortcuts and the ability to read and create virtual MIDI connections. For coding and productivity, user tests will give us insight in areas that would benefit those types of users on top of programmable hotkeys and auto-placement of windows. The final use case, gaming/streaming, will be modeled on the UI of the Elgato Streamdeck, with the programmable buttons acting as quick access keys.

CONTENTS

I. Introduction

Our sub-group has been tasked with taking the idea behind the Elgato stream deck to the Asus ZenBook Pro Duo. The Elgato Stream Deck is a hardware set of buttons that gives streamers access to commonly used actions. Our goal is to create a software version of this for the companion screen. We then plan to expand this application to interact and be useful in many other use cases including programming and music production. The end goal being to have a functional prototype to show how the companion screen could be utilized in such a way that increases users productivity.

This document goes through the components we plan to implement in the Asus ZenBook Pro Duo and how we plan to do so. We start off with the Dependency Viewpoints, which are the applications (music production and programming) that we plan to implement that would take advantage of both the companion screens and the additional functionalities we plan to provide. The next thing we go into is the Interaction Viewpoint which goes over how we want the system to remain consistent even though we are adding multiple components to it. Next up is the Interface Viewpoint that delves into how we are going to test the product in order to make sure we are still meeting our goals. Then to finish it off, we will go into the Structure Viewpoint, which takes a more in-depth look of the components we want to integrate in the ZenBook such as the use of custom application, integrating existing application, adding hotkeys, and along with the issue we hope to tackle that deals with returning focus to the main/companion screen from where the user was at previously.

II. Glossary

Term	**Definition**
Digital Audio Workstation (DAW)	A program used for producing music
Musical Instrument Digital Interface (MIDI)	Used for communicating between musical hardware
Integrated Development Environment (IDE)	A software application that enables programmers and developers to and develop software
Application Programming Interface (API)	A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service
User Interface (UI)	The design of the elements a user interacts with
User Experience (UX)	The overall set of elements that determine how well a user can interact with a software. This includes color choice, font choices, placement of elements, etc

III. Dependency Viewpoint 1: Music Production

To integrate our application with any DAW two main parts will have to be done. keyboard shortcuts, which will be set up the same as any other layout. The other being MIDI signals. This will make this use case a bit more challenging to implement. It will require emulating the support for MIDI which is generally a signal sent from the hardware. But with this set up it will let the user control almost anything in any DAW like they would with a hardware MIDI device. This includes controlling effects, controlling synth settings, and even playing notes.

A. Design Concerns

Since MIDI signals are traditionally sent using hardware we will have to emulate them. But because they are just bit values they can be represented in software pretty easily. The only problem is getting the signals from our software to the DAW. To do this it will involve using virtual MIDI connections. which are drivers that emulate the cables that would traditionally connect from the hardware to the computer.

B. Implementation

The way that MIDI signals work is that a hardware device, or software in our case, will send an 8-bit status signals that represent what note/control signal the device is controlling/sending as well as which MIDI channel to send that data on. Then two 8 bit data bits are sent that define aspects about the status. For example in the instance of sending a musical note the first data bit will state the pitch, or which note is being sent. And the second will tell what velocity, what volume/how hard, the note was hit.

For emulating the MIDI connections we can manually write our own windows divers but open source versions of these drivers already exist and can easily connect to our application. One that we will be using is called Springbeats. This driver works by creating eight virtual MIDI cables that can be used by any program on the computer but specifically the users DAW will use them to connect the same way a physical cable would. Our program just has to connect to it to send these values. At its basic set up we can set our program to send the signals and then use this as a middle man to connect to the DAW. Though this is a fine solution it would be nice to be able to incorporate the installation of this driver into our program.

IV. Dependency Viewpoint 2: Programming

A use case that we want to focus on is a use case for programming or other productivity use cases. In order to make this work, a section of this project will be geared towards making productivity as easy as possible. To outline the timeline of this section, we will be conducting user tests, then we can begin developing this section of the overall project. Finally, we need to create a good user experience for this section.

A. Design Concern

Part of increasing the user's productivity when utilizing the second screen is allowing some of the content to be on the companion screen in an accessible format. We were warned early on that the companion screen may not be at the best resolution to be able to not only allow selection but also reading. We will need to look at ways that would allow us to adjust the resolution automatically depending on the content that is then transferred to the companion screen.

B. User Feedback

The first step is recruiting people to do user tests in order to understand what to keep in mind during development. We can get a clearer picture of how the app should function and what specifically we can do to increase productivity. Finding and having these people give us feedback would happen around the first two weeks of Winter Term. Once we have user tests, we can begin developing this section of the app. This will begin about week three and will most likely last until Spring Term.

C. Implementation

C# will be the main driver of this section of the project, as much of this section will be high level. Creating a good user experience will allow us to deliver our ideas to them in a clean and approachable way. This part will be started alongside the development of the app, as we can begin drawing mockups based on the feedback from the user tests. Reviewing with these users periodically will allow us to alter our mockups to fit their ideas until we can present the app to them. This as well will mostly likely last until Spring Term. Then once the widget could independently work on the device, it could then be customized and integrated into specific applications with more tailored functionalities.

V. Interaction Viewpoint: Layout and User Interface

One of the major goals for the layout and user interface is that it fits seamlessly into the Windows operating system. We want to use consistent fonts and styles so the interface doesn't look out of place when the application is in use. Another goal is that the layout should be easy to use and understand. New users should be able to figure out how the application works without frustration. The interface is the link between the user and the application.

A. Design Concerns

Concerns about the layout and user interface mostly revolve around keeping everything consistent. When having a team create different sections of a project, we will have to be diligent in making sure we are following the same guidelines. Another concern is making sure the application doesn't seem out of place on the operating system. We want our application to feel native to the laptop, so designing with that in mind will help influence our design decisions.

B. Implementation

We can accomplish the first goal of consistency by taking advantage of Microsoft's style guides. Microsoft UI Fabric is a collection of UX frameworks that give developers the ability to build applications that fit in with Microsoft products. This package contains everything from colors and fonts to layouts and animations. Utilizing this tool will give us a good place to start when it comes to styling the user interface. Creating a straightforward layout that takes advantage of the companion screen and is easy to understand is the other challenge we must overcome. Using consistent input controls will help the user understand what each part of the application does. We want to give the user the ability to customize the icons of the buttons that they configure, much like the Elgato Streamdeck does. Any menus that we use should be styled the same with a similar layout so the whole application is consistent. In order to confirm that we have a satisfactory user interface, we will conduct user testing using mock ups and paper prototypes. This will allow us to iterate over the design and create something that people want to use.

VI. Interface Viewpoint: Software Testing

Once a prototype has successfully been built, it must go through a series of tests to make sure that the prototype is functioning properly without any obvious bugs. Software testing, as a concept, encapsulates most testing types; it evaluates and verifies that a software is bug-free, making sure that the software fulfills the requirements of the client, and passing boundary cases of a software effectively. Software testing has two different variations: black box testing and white box testing. The former tests for general functionality of the software without access to source code, while the latter tests the internal structure of the application to make sure everything runs smoothly. Since testing is done after or during development, software testing will not play much of a role right now. However, as developers, the preferred method of testing should be white box testing.

A. Implementation

In the actual process, the exact steps will largely be up to the user, but the main thing is that all intended functionalities of their prototype must be functional without any noticeable bugs. However, testing will largely depend on what language each group member will use to implement their prototype. In either case, the process will essentially be the same: test each feature of the prototype one by one; make changes to the code if things run as unexpected; rinse and repeat the process until all intended features are functioning properly.

VII. Interface Viewpoint 2: Music Production

While all our sections will have a similar UI to give the user a consistent experience. This section may differ slightly in some ways. The plan is to take inspiration from the design of the Abelton Push MIDI controller. The reason for this is two parts. One the Push is a device that is widely popular for music producers. Meaning it is a tried and true design, as well as an easy transition for many users. Two it the design is simple and not far off of what the Elgato Stream Deck looks like. Making it easy for us to keep a streamlined design between each use case.

A. Implementation

The way the Abelton Push works is it has a set of color-changing buttons that can be set to play sounds or control the DAW. These are laid out in a big grid in the middle of the device taking up most of its front. It also has some nobs and other buttons that have specific uses around the main button layout. For our application, we will use the idea of the colorful buttons. But also add on the ability for them to have to change text and icons behind them being that we have more freedom with a screen than a hardware button.

Beyond simply adding the buttons we can add software versions of the nobs as well. These will work by rotating your finger along the nobs. These will let the user emulate the hardware midi nobs on the device and control things like volume as well as synth settings.

VIII. Structure Viewpoint 1: Returning Focus

One issue that can easily become a problem when using a second screen is clicking something in the second app and then trying to use the original app again because the focus will have been changed to our application. We want, depending on the shortcut, to return the focus back to the application that the user is using.

A. Implementation

This can be solved using windows APIs, specifically from the Winuser.h library. Using these APIs our application has the ability to set which application is in focus on the screen. This is done by setting the z-order of the window. What the z-order is is a value that represents what order each window is on the screen, the higher the z value the closer to focus the application is. This is how using alt-tab to change to another application on windows works. Each application further to the right is further on the z-axis and has a higher z value.

For some applications, this won't be too hard because the short cut will correspond to the application we want to go to. For example, if the user was using our program for programming they could click a shortcut that swapped what tab they were in in the IDE/text editor. This would put our app in focus and then we would send the focus right back to the previous app. For this, we will use the SetFocus which gives focus to the application that we tell it to. This will work because we know exactly which application that the shortcut corresponds to. Other times it may be a bit more complected when the user clicks a shortcut that corresponds to another application than the one they had in focus before this will use a few more sets. For example while streaming a game the user clicks a short cut to mute their mic but they want to continue playing the game. In this case we have a few extra steps to get back to the application.

A function that will be useful here is SetWindowPos. This lets the developer set the exact position that the application will be set. It is not just limited to the z value it can also set the x and y values as well, though not necessarily for solving this problem the x and y values will be useful for other aspects of our project.

IX. Structure Viewpoint 2: Custom Applications

The end goal of this project is to be able to develop a software/plugin that is compatible with the Streamdeck. By doing so, the Streamdeck will able to use the developed software to display to both screens of the Zenbook Pro Duo laptop. However, the problem is deciding what custom application to build. For now, this is a challenge that myself and other group members have, since we do not know exactly what custom application we want to build yet. On the other hand, our group has essentially decided on what integrated development environment (IDE) to use: Microsoft Visual Studio. Due to the fact that the Zenbook Pro Duo laptop has Windows 10 built into it, we figured that Visual Studio would the obvious testing environment. Although other IDEs were mentioned - Atom, Eclipse, Netbeans, and etc. - Microsoft Visual Studio was chosen due to its obvious support with Windows and the fact that the IDE supports compilation of a variety of programming languages. The only problem is to determine how to build an application/plugin that supports the Streamdeck.

A. Implementation

The Elgato development website has provided a documentation of how to create a custom plugin. They have also detailed the architecture of the Streamdeck and how each plugin will interact with the Streamdeck. Not only that, the documentation also includes sample plugins that are built using different programming languages, such as Javascript, C++, and Objective C. One of the sample plugins is a memory game built using C++, so even certain games will be programmable in the Streamdeck. Overall, the tutorial and the sample plugins from the website will be a valuable resource for our group to develop our own prototypes. However, our group is also planning on creating a framework for the Streamdeck. This means that we will preset certain features on the Streamdeck to begin with, and then use each of our own prototypes to demonstrate our individual projects. To successfully develop a working prototype, we need to decide what programming language and what IDE should be used.

X. Structure Viewpoint 3: Integrating Existing Applications

This application should also be able to interact with existing applications as well. Giving the user the ability to send out a tweet, or launch an application, at the press of a button. When this application is first installed, we want to ensure the user already has applications that it can take advantage of, so by adding compatibility with common applications we can ensure every user can have some benefit from our project.

A. Design Concerns

We want to avoid copying the Elgato Streamdeck when it comes to use cases with other applications. The Streamdeck provides some excellent utility for certain use cases, but we will have much more freedom with our implementation because we are not limited to buttons. We also want to ensure that we have a broad range of applications that have default compatibility. If all of the default applications are geared towards a specific use case, we will be limiting our potential audience.

B. Implementation

The Elgato Streamdeck comes with some default applications already integrated, so we will have an idea of what can be done then proceed to expand on it. By surveying a broad range of potential users, we can create a list of applications to natively support that will reach as many consumers as possible. The best way to accomplish this is to use APIs provided by the applications. Different applications give access to different elements through APIs, so we will have to decide what features we want to integrate from each.

XI. Structural Viewpoint 4: Hot Keys

One popular feature of the Elgato Stream Deck is its use of hotkeys, seen on the buttons of the physical deck. The website then interacts with the device to add functionality to these buttons. For instance, a gamer may choose their buttons to correspond to certain moves. The button shortcuts saves time for the user by saving a series of actions in just a click. In a similar fashion, instead of having an external Stream Deck device, the second companion screen can be utilized as a container for these buttons. These buttons can then be customized to the environment that the user is in and then tailored to the application. For instance, a certain application could come with its own pre-set of hotkeys while the user could still also add their customized hot keys to the device.

The main point of the integration of hotkeys would be to speed up the time the user spends setting up. This could be a hotkey that is available right when the computer loads that the user just needs to click to boot the applications that they need for the day or could be a tool the user uses within another application, such as Twitch. Thus the first part of this task would be to create a hotkey-widget like application on the device. This would be done in .Net as it a framework specifically intended for the Microsoft framework.

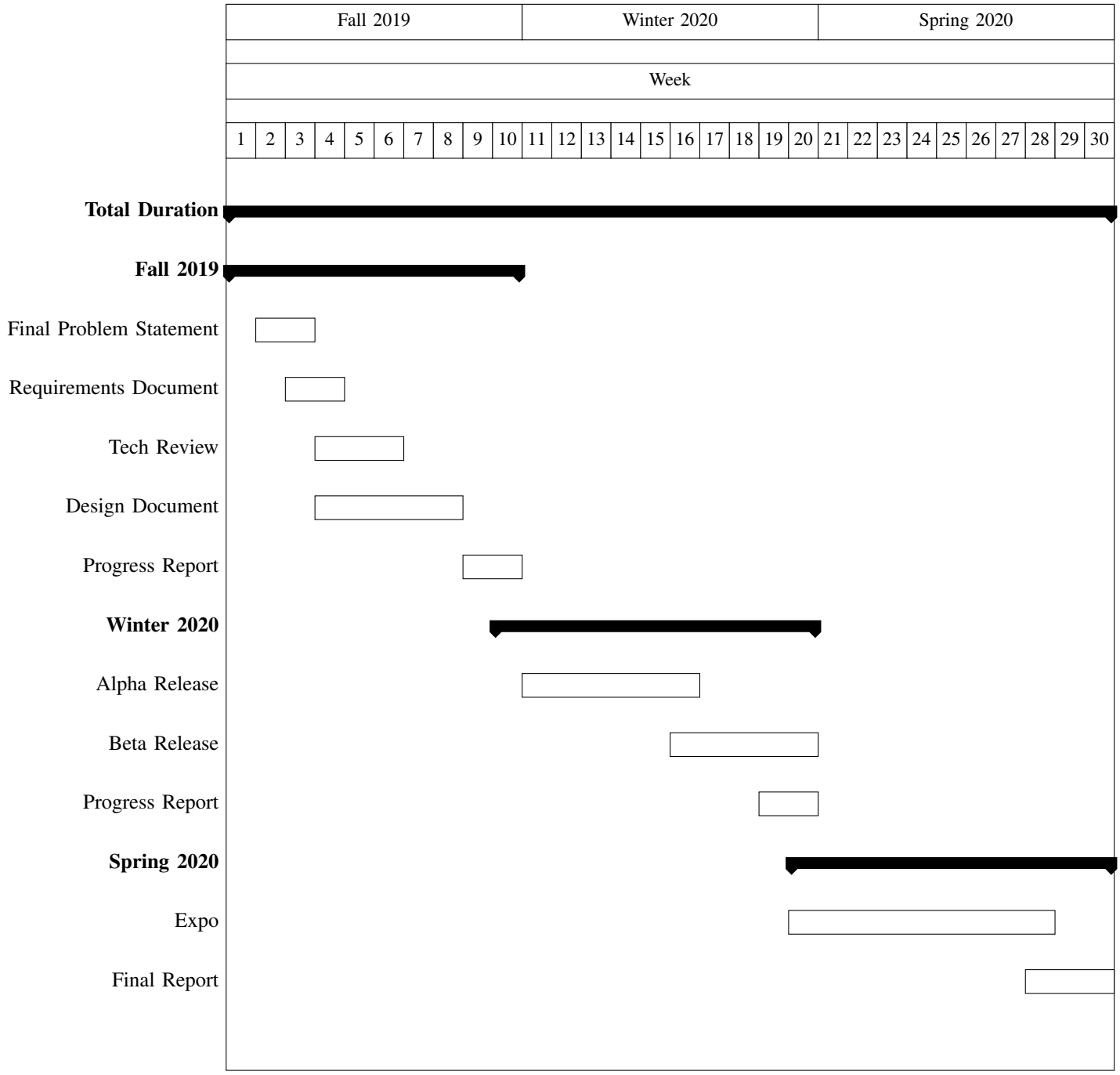
A. Design Concerns

One of the things to keep in mind here is accessibility. Ideally the hotkeys would be used from the companion screen by tapping on an image. However, we have been told that resolution may not be the best, making it harder to click on the icons. This would then pose a greater challenge to those visually impaired. Things to look at here would be adjusting the icons resolution, giving the user the option to adjust the size, or introducing a sound pertaining to a certain icon.

B. Implementation

Microsoft has its own software framework called .NET that basically builds programs written in a spectrum of languages such as C, C++, JavaScript, etc. Additionally, .NET also offers a bit of functionality to the framework such as Common Language Runtime, which provides services such as thread and memory management. This is especially necessary as once a user creates a hotkey, ideally it would be accessible at a later time. For instance, a hotkey could be used once the laptop starts up and so a user chooses all the programs they want booted through the corresponding hotkeys. Thus, the keys would then need to be properly stored in memory so that may be later retrieved even after the shutdown of the device. Additionally, a user should be able to create a hotkey that starts more than one processes.

XII. Design Timeline



XIII. Conclusion

Since the idea of two screens on one device is a relatively new concept that is now becoming more common, especially with future releases, our project problem originally started out as a way to utilize the companion screen to increase user's productivity. We have decided to take a model from the Elgato Stream Deck to build upon this idea. The Elgato Stream Deck is attractive to a spectrum of users because of the functionalities it provides. Our plan is to take those functionalities (returning focus and productivity) and then integrate them into both existing and custom applications. From there we plan on testing them with experienced users that represent the use cases in order to better adjust our tools. Above, we went into our main goals that we hope to hit; broken up into the different sections and how we currently plan on achieving them.