Expertise
and insight
for the future

Benard Gathimba

Windows Application Development: Desktop User Interface

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

8 March 2021

Author Title Number of Pages Date	Benard Gathimba Windows Application Development: Desktop User Interface 39 pages 8 March 2021
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Mobile Solutions
Instructors	Ari Puuskari, CEO Profox Oy Kari Salo, Principal Lecturer

This thesis project was commissioned by a Helsinki based company. The goal of the thesis was to create a Windows desktop UI application that would launch modules from a centralized UI. The purpose of the application was to improve the users' experience of the case company's software products. The development process of the application was segmented into two sections: The Research section and The Implementation section.

Section one involved a research of technologies and existing solutions that serve a similar purpose. Moreover, the research section highlights software development concepts such as the software integration architecture for both the Windows platform operating system (OS) and the software application. Section two involved the implementation process for the application based on the software requirement specifications.

The project resulted in the creation of a UI application that could launch multiple modules from a single interface. Further, the application was extensible allowing future addition of software module products. In addition, the application now provides an uncomplicated method of interaction with the case company's products by its customers. Finally, future studies to measure the application's impact to customer productivity could be carried out. The studies would highlight possible improvements to be made to the application.

Keywords	User Interface, Desktop Application, Windows, UI, C#, WPF

Contents

List of Abbreviations

3D	Three-dimensional. A perception of depth that represents objects or data.
IDE	Integrated Development Environment. An environment utilized for software development.
OS	Operating System. The system software that manages a computer's resources, programs and hardware.
PC	Personal Computer. A computing system for personal use.
SRS	Software Requirement Specification. A document that details the needs satisfied by a developed software product.
UI	User Interface. The means of interaction between a user and a system such as a computer, application or website.
WPF	Windows Presentation Foundation. A user interface framework utilized to create desktop applications.

1 Introduction

This thesis report was commissioned by Profox companies Oy, a Helsinki based company. The case company specializes in creating virtual designs for large scale construction projects. The virtual designs utilize new technologies to benefit a construction project in aspects such as project accuracy, financing, scheduling, collaboration and so forth. Moreover, one such technology example is the Navis Typhoon platform.

Navis Typhoon is a visual project portal that manages and shares a project's 3D model information, point cloud data and 360-degree images to all project stakeholders that include authorities and regulators, design consultants, architects, engineers, financers and so forth. In addition, Navis Typhoon currently utilizes four software modules namely: Admin module; Site control module; As-Built module; and Web module. Furthermore, the four modules are tools utilized to evaluate the project progress during its life cycle.

The idea for this report was conceived from the case company's need to improve the users' experience of customers utilizing Navis Typhoon. Currently, customers can only launch the different software modules separately. To improve usability experience, a viable solution would be creating a single User Interface (UI) with all product modules. Moreover, since the modules are developed for the Windows environment, the UI would also be a Windows application for the desktop Personal Computer (PC). The goal of the thesis project is to create a Windows desktop UI application that would launch modules from a centralized UI. The UI would also need to be dynamic to enable the addition of more modules in the future.

2 Project Theory

This chapter will be a theoretical review of software development. The section will analyze aspects such as software integration for the windows platform, the design and integration of a UI, a review of existing UI solutions that serve a similar purpose to that intended by the project and finally the development tools that will be utilized.

2.1 Software Integration

There are numerous integration procedures that can be adopted for a piece of software based on the context of use and the level of sophistication required. Moreover, such procedures vary based on aspects that include but are not limited to the operating system, target platform, hardware or software technologies. The first context to consider is that the case company's project should run on a Windows operating system. An Operating System (OS) is a piece of software that efficiently manages a computing system's resources, hardware and executes programs [1, p. 154]. The next context is the target platform which in this case is a Windows desktop PC and finally the subject of how to integrate the software for the chosen OS and platform.

Integration can be defined as the assembly of different components to produce one system or subsystem [2, p. 2]. A computing system contains various components that enable it to perform as expected. In addition, such components include the OS that allows a user's interaction with the system through input and output devices, namely audio devices such as a microphone, a display monitor, keyboard, wired or wireless networks and so forth. Moreover, the OS is divided into several layers that manage different aspects and functions that are a result of command prompts by the software or hardware. The induced command prompts can be categorized into four main sections.

The first section involves single-threaded applications that run exclusively over an underlying processor core layer. Therefore, this type of applications rely on embedded device drivers and libraries generally developed using the C programming language so as to access peripheral devices such as keyboard or display. The second section categorizes multi-threaded applications that are also executed over an underlying processor core. However, multithreaded applications require the use of a kernel or an OS that

would permit application switching so as to utilize multiprocessor communication layers and middleware. The third section entails hardware accelerators that are programmable to enable faster execution of tasks and functions. The programmability of the hardware enables the execution of smaller firmware programs which could receive software updates. Finally, the fourth section involves hardwired hardware accelerators. These hardwired accelerators have a set parameter configuration during assembly and cannot flexibly receive software updates. [3, pp. 90-91.] Figure 1 below is a diagrammatic representation of these four sections with abbreviations HW and SW for hardware and software respectively.

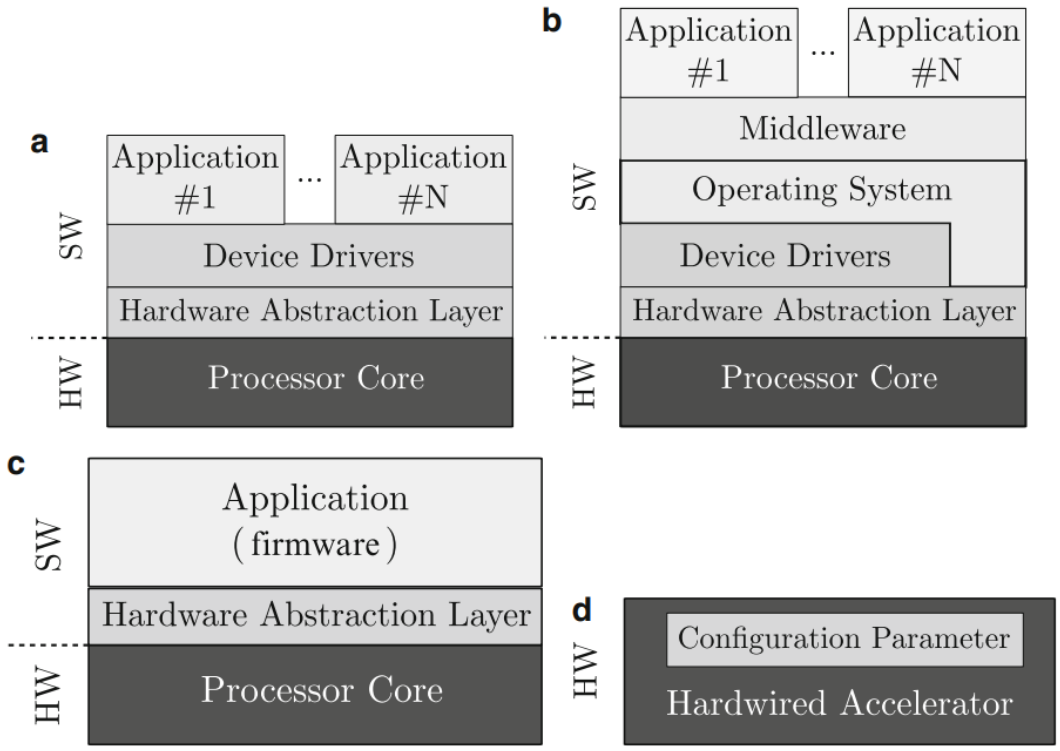

Figure 1. The software and hardware infrastructure model based on the source of the execution command prompts. a) Single-threaded applications. b) Multi-threaded applications. c) Programmable accelerators. d) Hardwired accelerators. Copied from Kempf et al (2011) [3, p. 91].

Figure 1 illustrates the dissected layers of interaction that combine to execute a command prompt sent by either the software application or hardware. The location of an application on the hierarchy stack denotes what underlying layer the application can immediately interact with. Figure 1 a) denotes applications that prompt device drivers such

as those utilized for a keyboard. Figure 1 b) denotes applications that require the middleware and OS to allow access for other functions such as multiprocessor communications between layers. Figure 1 c) illustrates applications that are responsible for hardware acceleration through firmware updates and finally Figure 1 d) shows a hardwired hardware accelerator whose configuration parameter values are set during manufacture and do not have a mechanism or application to update the configuration post-manufacture.

The model illustrated by Figure 1 b) would be a sufficient solution architecture for the case company's UI application. Typically, the model would enable the UI application to execute functions such as those in the following scenario: First, the application would use device drivers to send a launch command induced by a mouse click to the OS; next, the OS would execute the sent command by launching the correct application and designating the required resources to support it.

To implement the above scenario, the OS has to utilize components that create a suitable environment to run the UI application. The following are the components that an OS provides:

1. Program Process Management: The component responsible for tasks such as program creation, loading, optimization, execution, process scheduling, process synchronization, suspension or resumption and finally process termination.

2. Memory Management: The component that keeps track of the main memory utilized by application processes and determines whether to increase or decrease memory allocation based on program activity.

3. Storage Management: The management of secondary memory storage where application data is stored. The responsibility of the component is to manage file creation, deletion or manipulation by a program or the defragmentation, allocation or de-allocation of storage blocks.

4. Input / Output: The component that gives a level of abstraction from hardware devices, maintains the interfaces of device drivers, ensures that devices are utilized properly and avoids device errors.

5. Protection: The component that provides security to kernel code, hardware, data, files and processes from otherwise malicious or erroneous programs.

6. Networking: The component that enables processors to communicate with each other through communication lines called networks.

7. Command Interpreter: This component is responsible for facilitating the interaction between a user and the operating system through compatible environments. [4, pp. 41-42.]

To evaluate the components that the case client's project would need to integrate, it is important to know all the services offered by an OS. Figure 2 below depicts the services offered by an OS as a result of its components.

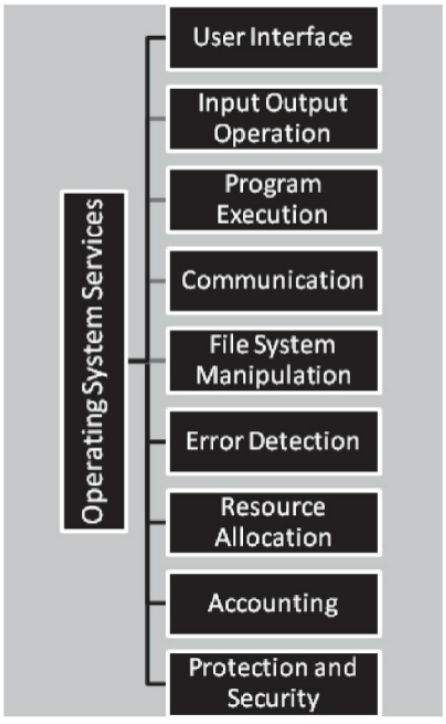

Figure 2. Services of an Operating System. Copied from Garg (2017) [4, p. 43].

Figure 2 above shows numerous component services offered by an OS to an end user. All the services are essential for adoption and integration during the development phase.

Moreover, the purpose of the services can easily be discerned from the OS components with the exception of accounting. In this context, the accounting service means the OS will track system activity for future performance optimization.

There are two programming models to consider during the development phase of the case project. The first model is the event-driven model in which the system handles the program using event interruptions triggered by either the software or hardware. The premise of this model is that there are two contextual channels. One channel is meant for the event handlers that are urgent and of a high-priority in nature while the other channel is meant for the flow of the order of instructions known as tasks. A suitable example of these two channels would be an event that triggers when a packet of data is available for processing. The event is handled using the first channel as it is time-critical though the processing and storage of the data in the packet is handled using the second channel as it may take additional time. This approach avails the system to new event triggers without blocking or queueing events while executing the tasks required by earlier events in a separate channel.

The second programming model is the thread-driven model. In this model, the system allows multiple program processes or threads to run concurrently on a single central processing unit. The main benefit of this model is that it leads to the creation of software components in an organized and logical manner as more emphasis is placed on the sequence of task executions. [5, pp. 699-700.]

2.2 User Interface Integration

This section will review the UI as one of the components in an application and the manner in which it is integrated. Further, an application is made up of various multi-layered components that interact and communicate with each other to perform the application's function. The number of layers as well as their functions vary depending on the purpose of the application and the target platform such as the Web, mobile phones, robots or machines and so forth. The report will make use of two applications to illustrate the different components utilized in creating the applications as well as show how the components are categorized. The first application is an employment information management system that is offered as a Web Service application. This application aggregates the information

of college employments provided by users with different roles such as guests, students, teachers and administrators. The Web Service application's framework design is shown in Figure 3 below with abbreviations DAL for the Data Access Layer and BLL denoting the Business Logic Layer.

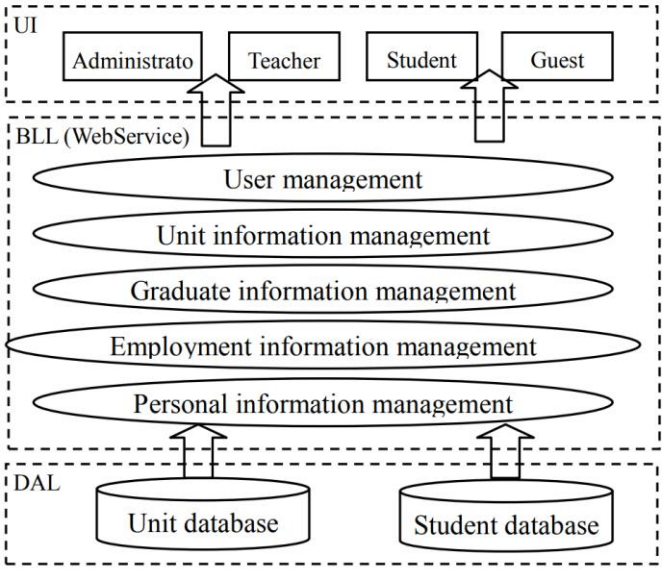

Figure 3. Web Service application for the employment information management system. Copied from Meng (2013) [6, p. 1933].

Figure 3 above illustrates the framework design for the Web Service application. The components in the UI or Presentation layer offer interactive interfaces for the users. The BLL serves as a link between DAL and the Presentation Layer and manages the efficient transfer of data between the two layers while also maintaining data security. The DAL provides a means for data manipulation and persistence storage. [6, pp. 1931-1933.]

The second application is called Yamasuta and it is a financial data analysis software robot. The robot is used to study financial markets and to provide users with investment decisions. Due to the volatility of financial markets, a financial software needs to analyze and process large data and to subsequently display the computed results dynamically through views and charts that can easily be understood by human end users. The components of the Yamasuta robot are structured into a seven layered model. However, the component-based architecture for an application comprises of the following overarching parent layers: The Presentation layer; The Application layer; The Integration layer; and

the Data layer. Figure 4 below illustrates the architectural layers of the Yamasuta platform application.

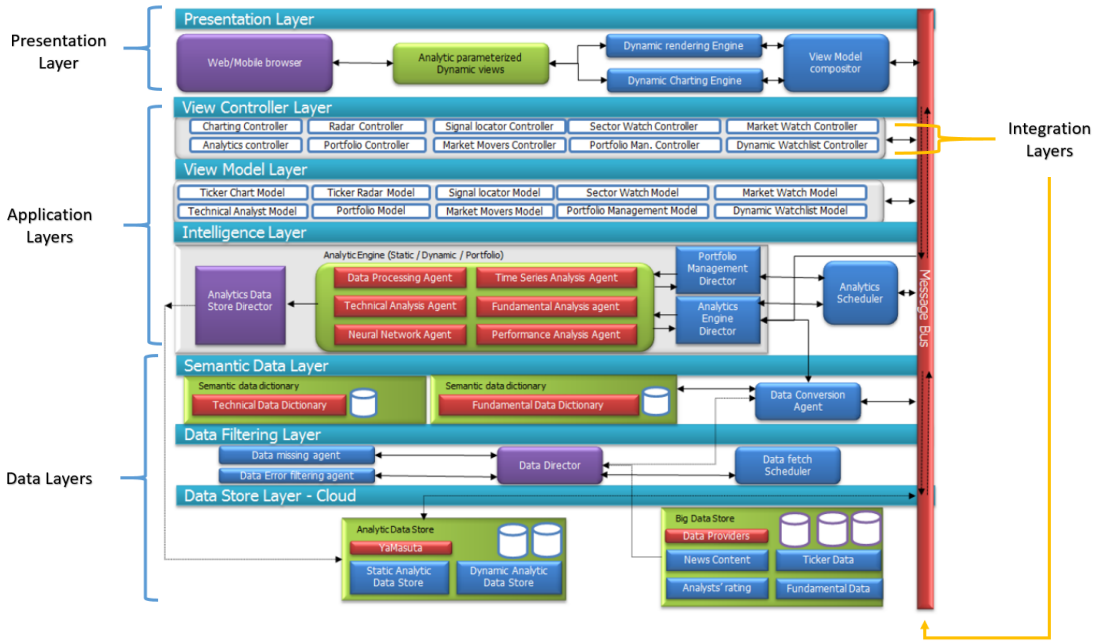

Figure 4. Framework layers of the Yamasuta platform. Adapted from Dayyani (2016) [7, p. 47].

As can be viewed from Figure 4, each major layer of the application has numerous components. Users interact with the Yamasuta application through the Presentation layer that is used to display content. The content displayed on the views varies dynamically based on the parameters set by the user. The Application layer is comprised of the view controllers, and the business logic. In this case, the business logic includes the view model and the intelligence layer. The Data layer is constituted by data storage and data manipulation components. Finally, the Integration layer denotes components used to facilitate the interpretation and communication between the four layers such as through the view controllers and the message bus. [7, pp. 45-51.]

The two software examples have similar framework layers with very different constituting components. The process of creating an application based on its framework and components should be efficient and reusable. To achieve this efficiency and reusability, a component library system is needed. This system is utilized to govern, detail, store and

fetch components for software development processes based on the software component reuse. Therefore, the component library system supports the entire software lifecycle from its development to its use. The component library is constructed hierarchically using a four-layer based architecture model that includes: The Framework layer; The Component layer; The Leaf class layer and; The Leaf method also known as Leaf function layer.

First, the Framework layer is used to define the main components of the overall software structure and their relationship. As a result, the assembly of components for each layer structure during development is directed by the framework layer. Second, the Component layer describe the actual components adopted for use during development that have a specific function such as project planning, design, analysis, test, document, data and so forth. Next, the Leaf class layer is the subject of object-oriented development used to define objects through classes and inheritance. The purpose of this layer is to increase the reusability of a component by other components or software systems. Finally, the Leaf function layer is the lowest layer of architecture and only provides services. This layer can further be divided into two categories comprising of the strategy functions and the implementation functions. The purpose of the strategy functions is to perform system-wide status checks in real-time, make decisions, manage resources, provide arguments and handle errors. The purpose of the implementation functions is to execute processes such as complex algorithms and return the execution status. A failed execution will have an error which is then handled by strategy functions. [8, pp. 101-103.]

The development process for a UI integration needs to be organized in such a way that the resulting application is robust and efficient. Consequently, the following software architecture elements are considered during the development process. The first element is the specifications which provide the scope for the basic concepts and restrictions for the application’s functionality. The second element involves the architectural components that resolve the specified technicalities. The third element involves the environment’s integration channels and internal integration frameworks that support the application’s architecture and components. The final element involves the architectural strategies utilized to implement the software system requirements.

The software architecture for an application should encompass the whole scope of infrastructure and levels where a user's functionality are deployed. As a result, the architectural elements exist in all levels of granularity that include the concepts, architectural patterns, strategies, components and relationships. [9, p. 370.]

To summarize the integration process for an application's software, consider the following hypothetical school application called Studies. Users of Studies include teachers and students with an account in the application. Further, students can log into the school system and access lecture materials from the application. Moreover, the Studies application can access the resources using a university's Web Application Processing Interface (API). The software development architecture for the Studies application can be found on Table 1 below with abbreviations SF and IF denoting strategy functions and implementation functions respectively.

Table 1. UI integration using a multi-layered multi-component based model.

<table>
<tr><th colspan="2">Framework Layer</th><th>Presentation Layer</th><th>Application Layer</th><th>Data Layer</th><th>Integration Layer</th></tr>
<tr><td colspan="2" rowspan="2">Composition Layer</td><td rowspan="2">Views</td><td>Business Logic</td><td rowspan="2">Database System</td><td>View Controller</td></tr>
<tr><td>View Model</td><td>Message Bus</td></tr>
<tr><td colspan="2" rowspan="3">Leaf Class Layer</td><td>Button</td><td>Web API</td><td rowspan="3">Database Connection class</td><td rowspan="3">Application or system dependencies such as a charting library, I/O system classes or Network protocol classes</td></tr>
<tr><td>Image</td><td>Student</td></tr>
<tr><td>Text Input</td><td>Teacher</td></tr>
<tr><td rowspan="2">Leaf Function / Leaf Method Layer</td><td>SF</td><td>Click method</td><td>API connection method. Responds with a new data packet</td><td>On start method</td><td>Open a connection method</td></tr>
<tr><td>IF</td><td>Connect to an API</td><td>Process the data packet</td><td>Create or Read from a database</td><td>Send or Receive data from an API</td></tr>
</table>

Table 1 illustrates the architectural structure and components of the hypothetical Studies application. The first column and first row of Table 1 represent the overall infrastructure

of the application. The other cells in the table represent sample components based on the architecture with subsequent lower levels of granularity. A use case scenario for the Studies application could involve the following steps: First, a student launches the Studies application which starts the on start method that ensures that a database system exists. Consequently, the user is then presented with a log in screen view that includes other components such as a text input field or a submit button. Further, the student enters the required credentials and presses the submit button that uses the click method to start a connection to the school API.

Next, the application's Business Logic determines which API should be accessed and what authorizations should be used. As a result, the integration layer utilizes an open connection method to send the credentials over a network protocol and receive data from the school API. Then, the business logic processes the data packet and displays it to the student. Finally, if the student choses to save the lecture material, another button click method is fired that opens a database connection and stores the data in the database system. This scenario shows how different layers perform different functions to meet the user's requirements while interacting with other layers and components. A similar architectural approach shall be considered during the development process of the client company's UI application.

2.3 UI Design

This section will analyze the components to consider when designing a layout. The design process of an effective UI layout should be centered and oriented to the user. Further, the design process should result in a product that achieves the intended requirements through a rigorous research of the user's psychology, behavior and expectation. [10, p. 171.] The meaning of an effective UI layout is that it should be one that functions as expected. To better understand user expectations, it is important to understand the users' needs. Research performed on users helps inform design decisions and results in a superior design product. The details of a UI design method that is user oriented can be found on Table 2 on the next page.

Table 2. User Centric UI design method. Copied from Liu (2016) [10, p. 171].

Category	Contents
Research	Define Objectives and Schedule
	Review current work and products
	Understanding user needs and behaviours
Modelling	User and customer model
Requirement definition	Scenario
	Describing the ability of the product
Framework definition	Defining information and functions
	Designing overall architecture of user experience
	Describing interaction of user roles and products
Design refinement	Making the design more specific
Support	Revising the design to adapt new constraints and time arrangements

Table 2 illustrates the different categories that constitute a design process. The research category refers to information gathering that instructs aspects such as the products objective, project schedule, existing solutions and user needs. Further, the modelling category refers to creating the product based on the user or customer. The requirement definition entails the product ability based on the context or environment used. The framework definition defines the development structure of the product, its functions and user interactions. In addition, the design refinement category refers to polishing the product functions and finally the support category adapts the products to changes or prior arrangements. Certain aspects of the modelling category will be discussed in subsequent subheadings.

2.3.1 UI elements

The UI is a link between human and system interactions [11, p. 2]. The interactions are categorized as either an inputs or an output. To adequately support input operations, the

UI has to present a structured layout comprising of elements such as icons, typography, animations and so forth. The elements create an environment in which the goal of the application can be realized for a user's needs based on the application's capabilities or limitations. The output operation is the presentation of system computational result. The presentation is done through UI elements in a manner that is understandable to the user. The following subheadings present a comprehensive review of some of the elements.

Layout element

The layout consists of multiple components that have been systematically arranged relative to each other to create a holistic view. The components include:

- Elements that perform tasks such as data analysis.
- Component elements that are based on the context of the state of the system, application or use.
- Elements that perform user's tasks and actions.
- The elements that enable the visualization of the interface presented to the user.
- The elements that result in the transformation of the layout. [12, pp. 8-9.]

A simpler explanation for a layout element is that items in a layout have a wide range of functions. For example, some items are responsible for input operations such as a text input element. In addition, an item such as an image element is responsible for visually presenting images to the user. Further, there are items responsible for performing user tasks or data tasks for example a button element for the former and a grid view element for the later. The grid view element can be used to display information from the database which is a data analysis task.

The project may adopt a more conventional rectangular layout design or experiment with more modern circular layouts. The effectiveness of the layout would be tested based on factors such as the application's ability to achieve its development objective and meet the user's needs and expectations.

Icon element

An icon is a picture that is used to convey the meaning of a word or idea in a piece of software system. Icons have been adopted for use due to the fact that they are simple to understand and are a communication language that is globally recognizable. [13, p. 954.] An example of an icon is the “X” image that is used to replace the word close and is understood to mean shut down the application. Similar icons will be adopted for the project as expected and the testing for the same would be in the ability for the users’ to understand the information conveyed by the icons.

Typography element

Typography is the method used to order characters such as alphabetic letters, numbers, glyphs, scripts, ascenders, descenders and so forth. The ordered characters make words or sentences that are printed in blocks known as a type. The purpose of typography is to make the written type readable, legible and appealing when printed or displayed on a screen. [14, p. 1.] For this project, typography would be utilized to determine the optimal design for the typography variables used on the type such as font size, weight, spacing, slant, kerning and so forth. A measure of the user’s satisfaction with the typography used can be evaluated using two category metrics: the objective metric and the subjective metric. The objective metric involves aspects such as the user’s fixation duration on the type used, the apprehension score, reading time and so forth. The subjective metric is mainly based on the users rating of the overall satisfaction with the typography utilized. [15, p. 2.]

Animation element

An animation is the result of playing successive image frames of an object with sufficient speed to create the illusion of movement. In digital products, the benefits of animations include making information interesting, brand recognition using memory retention and improved user experience. [16, pp. 376-377.] UI animations of an object may include any number of the following actions:

- Object scaling and resizing.
- Object translation from one position to another.
- Object visibility or transparency change.
- Object rotation. [17, p. 357.]

Color scheme element

Color can be described as the property of light that is visual in nature but not related to the light's glossiness, lightness, texture, saturation or translucency [18, p. 3]. Color can be classified into numerous existing theories such as the primary colors, secondary, complementary, tertiary, chromatographic, contrast and so forth. The ability to build effective digital media relies on the designer's ability to apply color harmony to an application. [19, pp. 1-14.]

2.3.2 Usability and user experience

The measure of usability on an interface is the degree to which the product, service or system is utilized within the specified context based on attributes such as learnability, effectiveness, efficiency, memorability and satisfaction. Usability is tested by focusing on how an interface can perform a user's needs based on interactive design or empirical calculations. [20, p. 146.] User experience is characterized by the following three factors: It involves a user; the user interacts with the interface of a product, service or system; the user's experience during the interaction is observable and measurable [21, p. 4]. The usability and the user experience of an interface are important aspects to consider during software development. The purpose of these aspects is to ensure the creation of functionally practical solutions that are enjoyable to use.

2.3.3 Accessibility

An application is designed with target users in mind. However, most applications fail to extend the user base so as to reach users with a limited dexterity, diminished vision, impaired speech or cognition. In essence, the purpose of accessibility is to increase a

product's distribution to reach additional users that are affected by a temporary or permanent limitation on their physical or biological abilities. Accessibility requires awareness of accessibility demands by application designers, developers and testers. [22, p. 94; 23, pp. 1-2.]

2.3.4 Functionality

There are software applications that are designed to offer services through user interactions. Such applications have two aspects that characterize their implementation. The aspects are the UI and functionality. When an application is launched, it starts a design process that is utilized by the two aspect to convey important information. This design process is known as a task model. The task model describes an application's interactive UI tasks in parallel to its functional or system tasks with regard to their interactions and temporal order. While the task model may function as expected during development, certain considerations have to be made. These considerations include: How the software will be utilized; the context of use and; the existing environment where the software is deployed. Thus, the developer is challenged to implement safeguards that guarantee that the software will function as expected. [24, pp. 2-10.] The safeguards may involve features such as validity checks for text inputs, size limits for resources, the minimum required version of dependencies and so forth.

2.4 Existing UI Solutions

This section will review existing UI solution from companies such as Adobe, Autodesk and Microsoft. These companies offer numerous software products to their customers packaged in suites. The section will study the different considerations made by each company during the development of an application manager UI. Moreover, the study will revolve around aspects such as the UI layout, functionality and usability.

Adobe Creative Cloud Application

Adobe is a computer software company that was founded in America in 1982. The company has numerous multimedia software products and services for creative design, photography, web, video and so forth. Customers of Adobe can access software tools through an online subscription-based payment plan, known as Creative Cloud (CC) [25, p. 22]. Furthermore, CC is a collection of software products that include Photoshop, Illustrator, InDesign, Premiere Pro, XD and so forth. Figure 5 below depicts the Creative Cloud UI with key features in the layout highlighted.

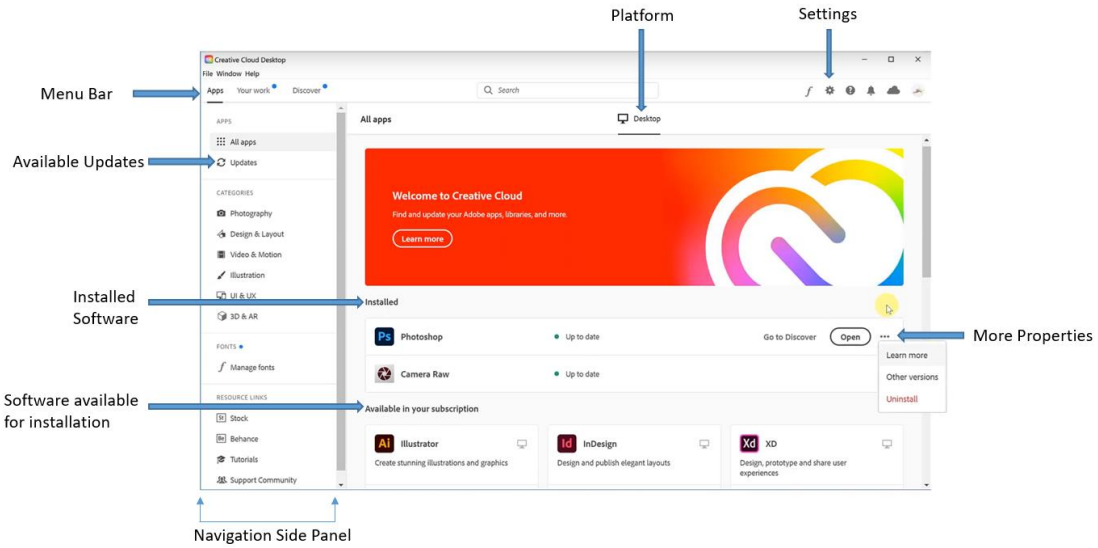

Figure 5. Creative Cloud user interface.

Figure 5 above illustrates the Creative Cloud UI. The UI has features such as the menu bar, the platform, a navigation side panel and so forth. This UI provides the users with an overview of all CC products as well as the ability to perform certain functions such as software updates, downloads or uninstallation.

Autodesk Desktop Application

Autodesk Incorporated is an American software company that was founded in 1982. Autodesk software tools are used by customers from professions that include but are not limited to media, manufacturing, construction, architecture and so forth. The customers use the tools to plan, envision and simulate various concepts. [26.] Autodesk customers

manage software tools under subscription through the use of Autodesk desktop app. The Autodesk desktop app offers customers training content and product updates based on purchased subscriptions. [27.] Figure 6 below depicts the layout of an Autodesk desktop app.

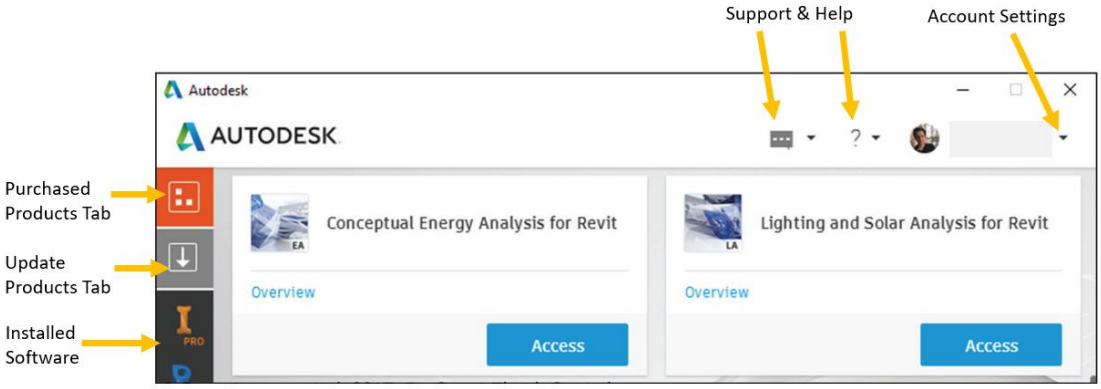

Figure 6. Autodesk Desktop Application user interface. Adapted from Kirby et al (2017) [28, p.928].

Figure 6 highlights the aspects used to interact with the Autodesk desktop app. Furthermore, customers can access or update software tools through tabs as shown in Figure 6. Moreover, customers can login to the application to activate subscriptions or update purchased software. In addition, access to learning content is based on purchased subscriptions. The purpose of the learning content is to empower the customers with knowledge of newly deployed features. The new features are a result of software updates that improve the product or patch any bugs.

Microsoft Office 365

Microsoft Corporation is an American company founded in 1975. The company provides numerous products and services such as electronics, cloud hosting, operating system software and so forth. Further, Microsoft offers Microsoft Office 365 which is a subscription-based application manager. Benefits to Office 365 subscribers include access to Office software, Office on Demand, software upgrades, SkyDrive online storage and so forth. [29, p.1.] Figure 7 on the next page shows the layout of Microsoft Office 365 application.

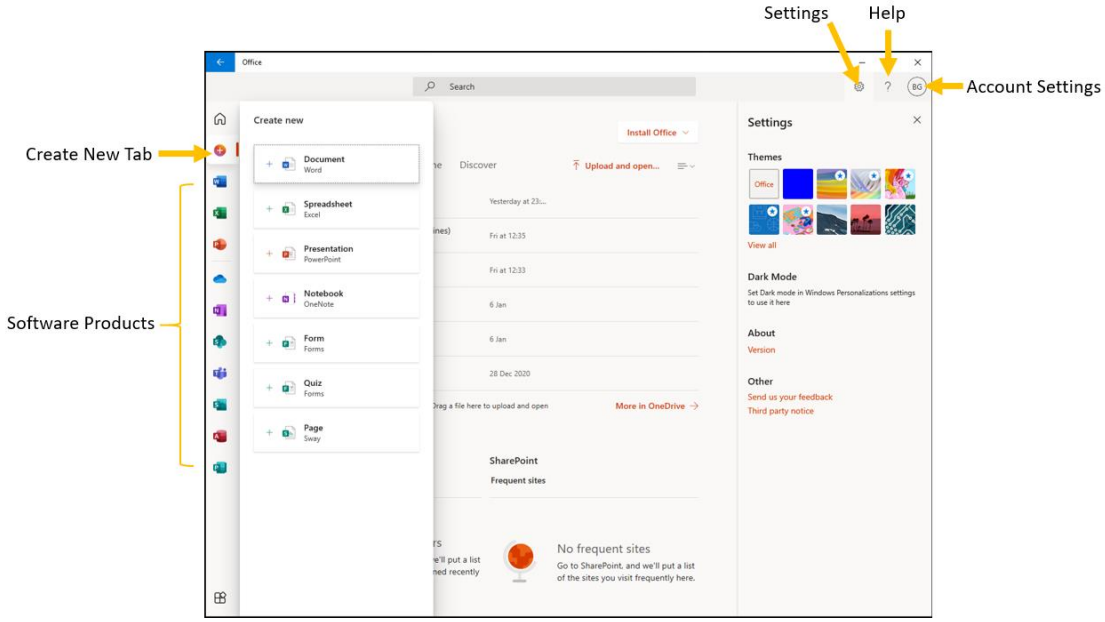

Figure 7. Microsoft Office 365 Application user interface.

Figure 7 above, depicts the layout of Microsoft Office 365. From the layout, the settings tab has a brighter background highlight as compared to the help and account tabs. The highlight is utilized to visually interpret an active tab. The Create New Tab has a similar highlight which infers that a consistent design pattern was utilized throughout the layout. Moreover, the layout includes numerous software products that are aligned on the left-hand side panel.

There are similar layout design aspects deduced from Figures 5, 6 and 7 that can be considered as a basic layout structure for the case company's project. The first aspect is that the layouts have the tabs that perform default Window operations to minimize, maximize or close the application Window. The second aspect is that the layouts have two major rows. The first row has been utilized as a menu bar and includes features such as the company logo, a search bar, settings tab, help tab and an account tab. The menu bar's function is to personalize the layout through account setup or settings, perform quick search operations, seek help from professionals or even give feedback. A major benefit of personalized UI layout is the improved user experience.

The second row is divided into two or three column segments. The first column is the left hand side panel used to navigate through software products. The second column is in

the middle and is the largest segment. This segment may contain more information about the installed software or software available for download. Moreover, the segment may contain useful information such as training or feedback. Finally, the third column is the right hand side panel that is used to display various aspects of features found under the settings or account tabs.

There is a logical business decision for companies to offer customers software tools and benefits such as software upgrades and updates. The decision is meant to ensure a steady business growth through the study of predictive user intended actions. A user intended action includes customer behavior patterns that are valuable to the company such as customer acquisition or loss, product subscription, defaults in payment and so forth. Through predicting customer intended actions, companies can enhance their business growth result. [30, p. 487.] Similarly, the case company seeks to proactively improve the customer experience by offering a UI application that manages all the company's software tools.

2.5 Development tools

This section will review a few of the development tools needed to implement a UI software. Moreover, such tools include the .NET Core, development language, Visual Studio, Blend, Project management tools and so forth. The review is meant to highlight the tools' contributions during a project development.

.NET Core

.NET Core is an open source cross-platform version of the .NET platform. It is composed of the following four main parts: .NET Core runtime; framework libraries; SDK tools and; language compilers. The goals of .NET Core include:

- The ability to support cross-platform development. In addition to the support of iOS and Android deployment through the use of Xamarin, .NET Core also runs on Windows, mac OS and Linux.
- Top performance: .NET core has a consistently high performance with additional improvements after each new release.

- A .NET standard specification for uniform portable class libraries that can be utilized across all .NET runtimes.
- Stand-alone or portable application deployment. This means that applications can either use the system-wide .NET Core installation or be published together with the framework.
- One of the primary objectives of .NET Core is to target command line support.
- .NET Core including its documentation are fully open source.
- .NET Core allows for interoperability with the .NET Framework. This means that it is possible to reference .NET Framework libraries for use in .NET Core. [31, pp. 1245-1246.]

Microsoft Visual Studio and Blend

Visual Studio is an integrated development environment (IDE) with tools for efficient and productive software development. An application may contain one or more project code. The project code is contained in a solution. The solution can contain projects that have been written using a mixture of .NET languages such as a C# project with a VB.NET project. Some of the tools that Visual Studio offers developers include a code editor, code navigation capabilities, debugger, testing, source control features and so forth. [32, pp. 9-210.] Blend for Visual Studio is a tool used as an interactive design tool for UI elements. It is utilized to build interfaces using features such as its graphical oriented tools, animation elements, visual template customizations, action triggers, effects, element binding and reactions to state changes. Further, Visual Studio and Blend for Visual Studio do not require extra configurations to allow for the synchronization of project files during development. [33, p. 449.]

Languages

Common Language Runtime (CLR) is a central component of the .NET Framework runtime execution environment. Code running on .NET runtime is often referred to as managed code. However, before managed code can execute on the .NET, it has to be compiled using the following two steps: First, the source code is compiled into Microsoft Intermediate Language (IL); second, the IL is compiled by the CLR into platform-specific code. The IL is a low-level byte code language that is based on numeric code and can be translated quickly into native machine code. In addition, the IL offers three distinct

advantages for development in .NET that include: Language interoperability; performance improvements and; Platform independence.

Language interoperability means that source code from one language can be compiled to IL and should then be interoperable to other IL code from other source code languages. Languages supported by .NET include C#, F#, Visual Basic (VB) and so forth. Performance improvement implies that IL performs Just-In-Time (JIT) compilation of an application's source code during execution while platform independence means that cross-platform implementation of .NET is possible. [34, pp. 3-6.] Figure 8 below illustrates a diagrammatic representation of the .NET Framework showing the different layers of infrastructure.

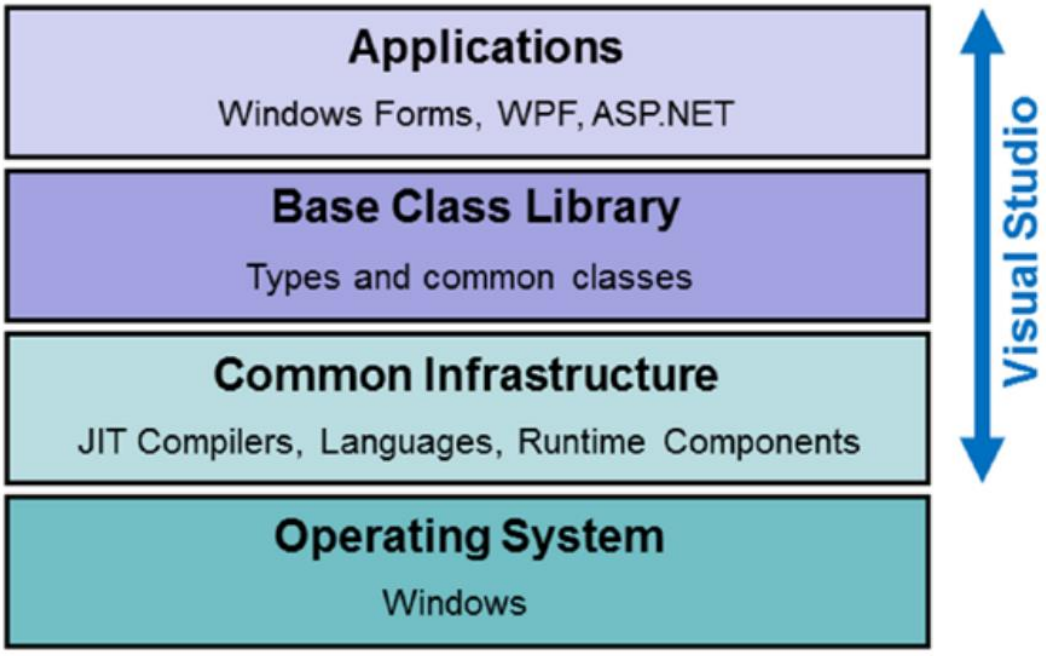

Figure 8. Infrastructure of the .NET Framework. Copied from Joshi (2017) [35, p. 11].

Figure 8 above shows the stacked infrastructure of the .NET Framework. Further, combining the literature from previous sections on .NET Core and Microsoft Visual Studio and Blend sections, it is possible to deduce the following: An example of a Windows Forms application written in C# can reference a VB base class library; the application's source code would then be compiled through the common infrastructure layer on runtime and executed by the OS in native machine code. In addition, the diagram illustrates the

difference between the .NET Framework and .NET Core in that .NET Framework is meant for the Windows OS while the .NET Core supports cross-platform implementation.

Source Control

Source Control Management is the process of making a catalogue of any modifications or customization to the source code of an application once it has been deployed. Modifications imply that the version of the current application is different from the previous application. There are many reason for creating new software versions such as for security updates or for integrating new features. However, such changes could introduce potential errors or bugs to the application. The purpose of version control is to enable developers to easily identify the cause of such errors and fix them. An example of a decentralized version control system tool is Git. Git enables developers to create repositories of their projects and store the project files on a local machine or on the server. In addition, Git gives developers access to operations such as file commits, view of file changes from history, revision control, continuous integration and so forth. [36, pp. 219-220.]

Entity Framework

Entity Framework is a component of the ADO.NET framework which is a data access technology for Microsoft .NET. Further, Entity framework is a bridge between practical applications and their database concepts. An Entity Data Model (EDM) incorporates relational models with the concepts of Entity-Relationship Model. In addition, some of the features of the Entity framework include: It provides an interface for EDM queries; can bridge from conceptual level to relational level using a mapping mechanism and; can extend relational data model. Data manipulation on the Entity framework are done using either one of the following methods: the LINQ to Entities; Entity SQL or; Query Builder. [37, p. 94.]

3 Project Implementation

This chapter will detail the development process during the implementation of the case company's project. The section will analyze topics such as the project objectives, software requirements, App design and creation, testing and publishing. The implementation of the project will be based on the theoretical concepts discussed in Chapter 2.

3.1 Objectives

The objective of the project is to create a user interface that supports the following aspects:

- The application should contain all the software modules sold by the case company in one centralized UI.
- The application should allow users to launch the software modules from the UI.
- The application should be extensible for future software module products from the case company.

3.2 Software Requirements Specification

This section will discuss the requirements that need to be fulfilled by the software. The software requirement specification (SRS) document is the framework by which a new piece of software is developed. Moreover, the document serves as an anchor used to verify and validate that the software satisfies the intended essence of the product. As such, the finished software product is put through qualification testing. The testing compares the developed software versus the SRS to determine whether the acceptable criteria for a desired system behavior has been met. [38, p. 33.]

In simpler terms, requirement specifications are an elaborate plan that consists of design, features, components and solutions. An effective method used to capture the essence of the requirements is known as "user stories". Figure 9 on the next page highlights the three components of a user story.

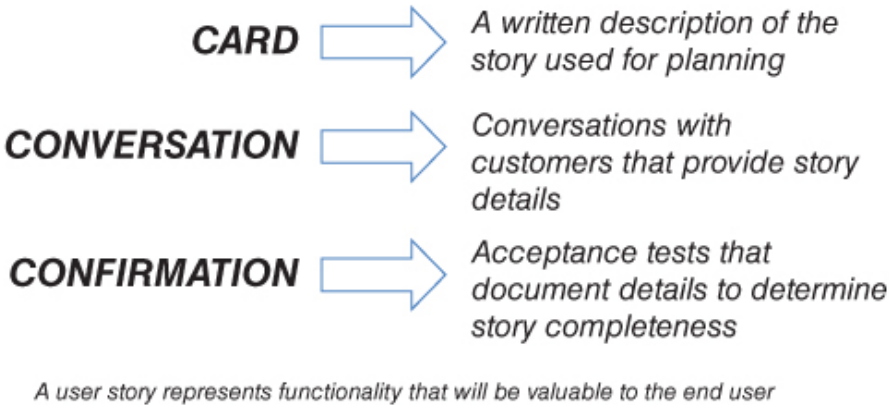

Figure 9. The three elements of a user story. Copied from Rachel (2015) [39, p. 91].

As illustrated in Figure 9, a user story has three components that constitute the functionality valuable to the software user. The components create a technical analysis for hypothesized requirements. Furthermore, the components serve as a template that incorporates: The user role, the goal to be achieved, and the benefit to be gained. [39, pp. 91-97.] This project adopted a modified version of the user story template to outline the software requirements. The list of requirements includes:

1. The application shall allow users to access all modules from one UI.
2. The application design shall have considerations for readability and usability.
3. The application shall have notifications that inform users of important status changes.
4. The application shall have the Web Module as the default module for all users.
5. The application shall have a database that stores all module data.
6. The application shall allow users to launch a module from the UI.
7. The application shall allow users to add modules to the UI.
8. The application shall allow users to scroll through a list of all added modules.
9. The application shall allow users to set some modules as active from a list of other possible modules.

10. The application shall allow users to update existing module data in the database.

11. The application shall have an adaptive UI that automatically changes to reflect the module updates without a need for a restart.

12. The application shall allow users to delete modules.

13. The deployed application should include all required dependencies.

14. The application should run on a Windows 64-bit Operating System (OS) with an x64-based processor.

3.3 App Design

The application was designed using an interactive design tool called Figma. During the design process, the following considerations were made: The UI would adopt either a rectangular or circular layout; the design should have a menu bar; the menu bar shall have buttons for Windows functions such as minimize, maximize and close; the UI should have a settings button; the UI should include a settings interface UI; the settings UI shall have a layout for module operations such as create, read, update or deletion of modules and so forth. Figure10 below shows earlier design concepts that were presented to the case company for consideration.

Figure 10. Project layout design concepts.

Figure 10 from the previous page illustrates two layout concepts with software modules aligned to either the left side panel or radially from a circular center. The case company chose the circular layout and suggested utilizing the rectangular layout for the settings page.

3.4 App Creation

The process of creating the application began with a plan on the logical flow of functions. Based on the case company's needs, the project would need to have a dynamic UI that would reflect changes to the added modules. Further, this meant that the application would need to have a database system that would store module data such as the full path to the module location. Figure 11 below shows the logical flow of operations on the application.

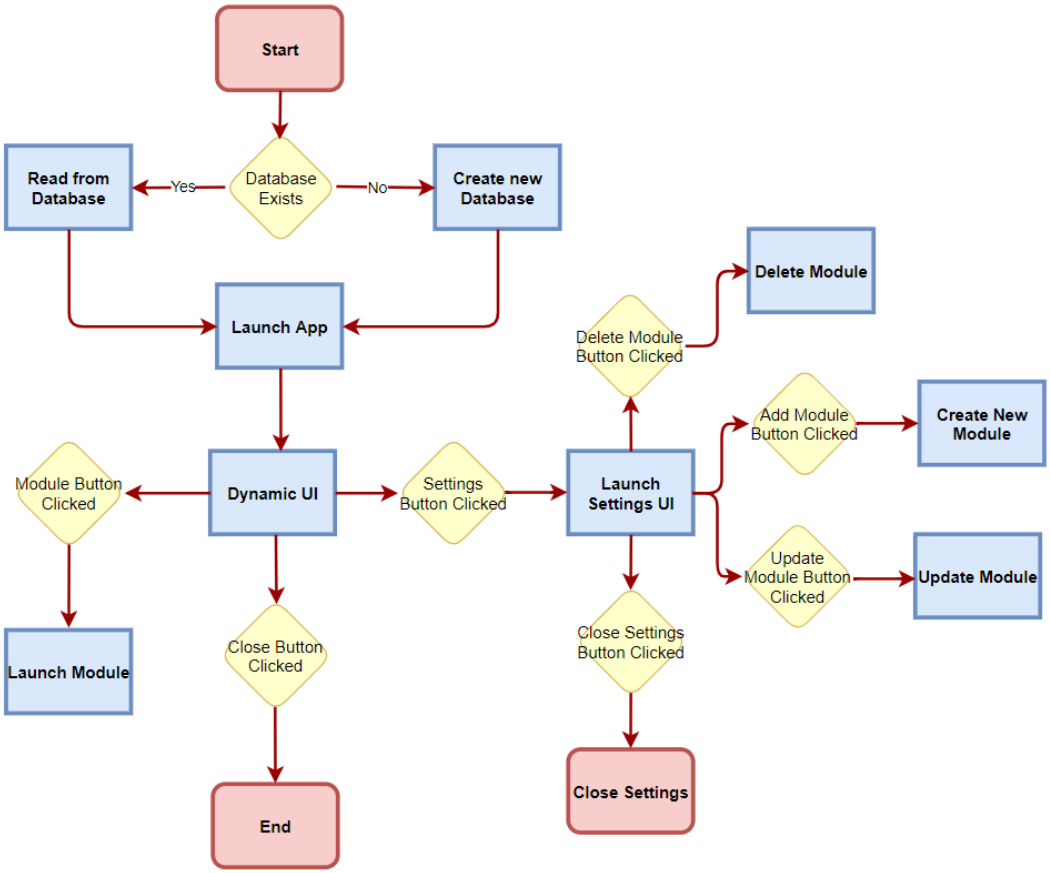

Figure 11. Project layout design concepts.

As can be viewed from Figure 11 on the previous page, the application performs numerous functions based on user interactions. Further, the application would need to be extensible to allow the addition of extra modules. Currently, the case company has four modules with the Web module as the default module for all customers. Therefore, the application should have a dynamic UI that would only display the Web module on initial launch and then change based on additional active modules chosen by the users.

The project was implemented using technologies such as: Git for source control and project management, Visual Studio IDE, Microsoft Blend for the UI design, SQLite database system using the Entity Framework, the Model-View-ViewModel (MVVM) architectural pattern and so forth. Figure 12 below shows a snapshot of the Kanban project management feature on Git.

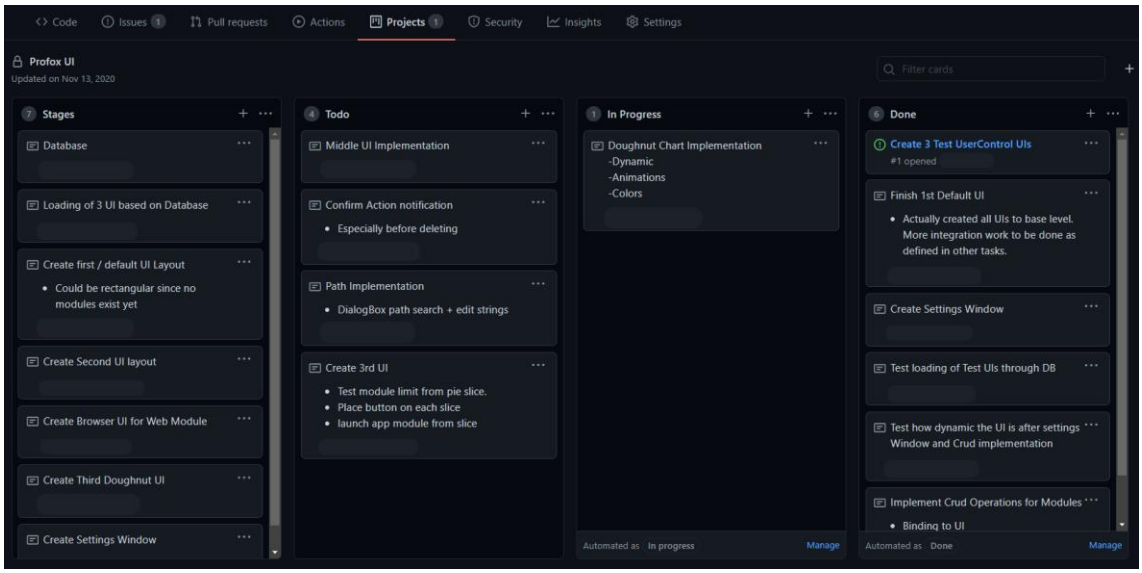

Figure 12. Git project management.

As can be viewed on Figure 12 above, the project was segmented into smaller sections. The advantage of utilizing this development approach is the ability to add new features in small increments making it easier to detect possible source code errors. The small segments are also incentives for development as each milestone completed serves as a reward for the overall project completion.

Visual Studio was used as the development IDE. The project folder structure utilized the MVVM model structure. The application's business logic and the database access services were separated from the UI views. The ViewModels were used as a bridge between the business logic and views. Figure 13 below shows the project's folder structure.

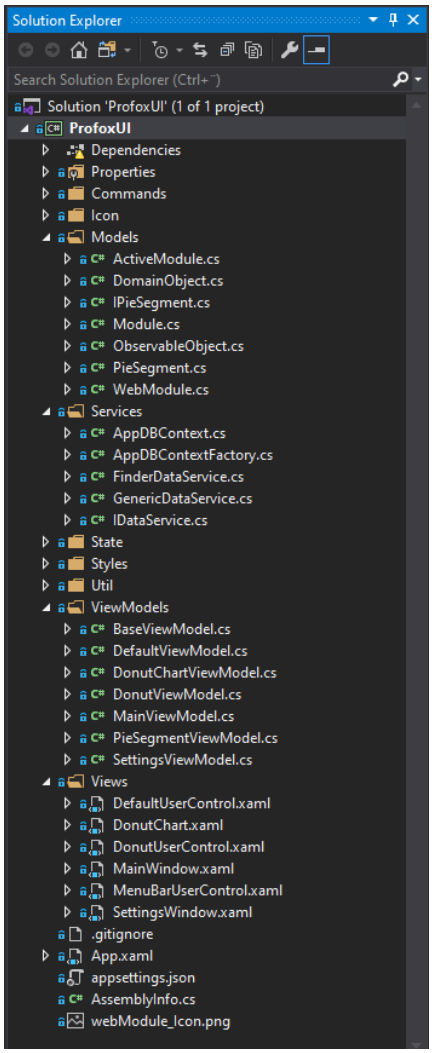

Figure 13. The UI project folder structure.

Figure 13 shows the directory structure of the application UI project. The project has a default UI that is displayed when the project is launched. Further, when a user activates additional modules, the default UI is replaced with a doughnut chart UI. The doughnut chart is segmented based on the software modules that are set as active. The changes in the state of the modules is dynamically reflected on the UI views displayed. The layout

for the UIs was mainly developed using Microsoft Blend. Figure 14 below shows the view of Microsoft blend.

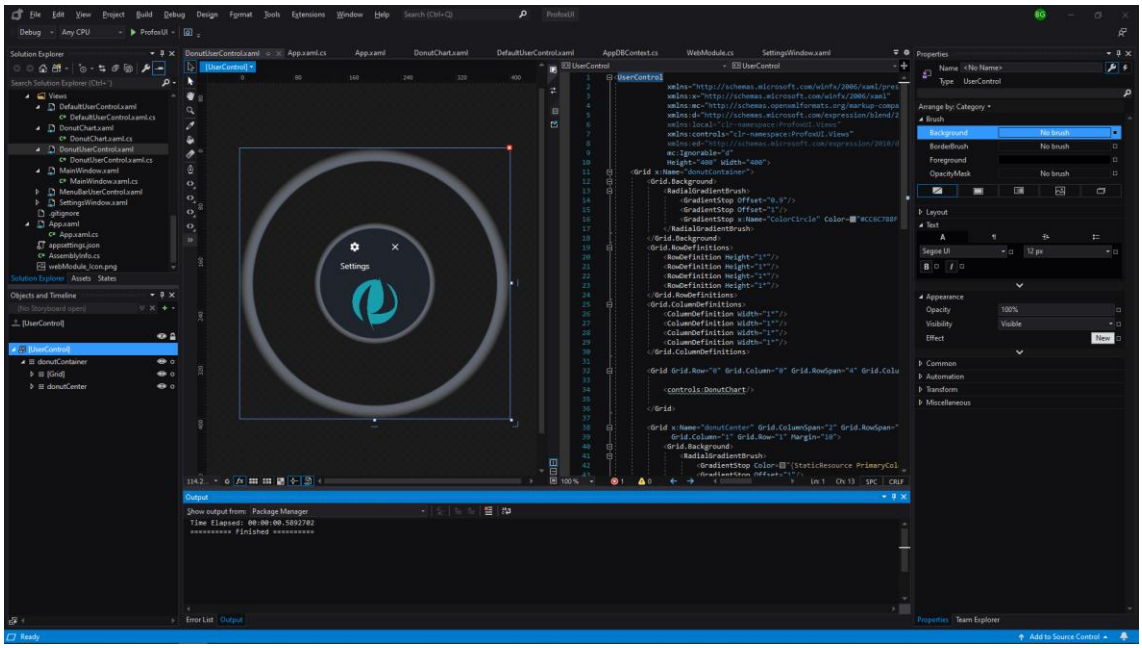

Figure 14. Microsoft Blend UI design tool.

Figure 14 above shows the design process for the doughnut UI. Blend creates UI designs and enables synchronization with Visual Studio. The arc segments for the software modules was created using Microsoft Expression drawing package. The project was developed using .NET Core and the Windows Presentation Foundation (WPF) which renders UI views.

C# was used as the development language for the project. The database was designed using the object oriented programing concept. In addition, the Entity data model allowed the incorporation of relational models between the module object and the active module object. All the modules were stored as a module object to the database. The active module object only stored modules that a user had activated. When the application is launched, the software retrieves all the active module objects if they exist and determines the type of UI to be rendered. In the event no active modules are stored, the application launches using the default UI. While the application is running, the active module collection is set as an observable collection and any changes to the size of the collection triggers a re-render of the UI. The UI is bound to the ViewModel using a TwoWay data

binding system that triggers the property change notification for any changes in the observable object collections. Figure 15 below illustrates sample source code for the main ViewModel.

```
2 references
public void Init()
{
    // Read DB values if any exist
    List<Module> modCol = _moduleService.GetAll().Result.ToList();
    List<ActiveModule> activeModCol = _activeModuleService.GetAll().Result.ToList();

    ModuleCollection = new ObservableCollection<Module>(modCol);
    ActiveModuleCollection = new ObservableCollection<ActiveModule>(activeModCol);

    if (activeModCol.Count() == 0)
    {
        UpdateViewCommand.Execute(ViewType.DefaultUC);

        // Update DefaultUI
        if (DefaultViewModel.getInstance() != null)
        {
            _defaultVM = DefaultViewModel.getInstance();
            _defaultVM.Init();
        }
    }
    else
    {
        UpdateViewCommand.Execute(ViewType.DonutUC);

        // Update DonutChart when changes happen
        if (DonutChartViewModel.getInstance() != null)
        {
            _donutChartVM = DonutChartViewModel.getInstance();
            _donutChartVM.CreateCollections();
        }
    }

}
}
```

Figure 15. The main ViewModel source code.

Figure 15 above shows the sample source code for the main ViewModel. From the code, it is possible to deduce that the modules and active modules are stored into separate observable collections. Changes in the observable collection will result in a property changed notification.

During the development process, the developer and the case company held multiple virtual meetings to discuss the project. The discussions of the meetings included topics such as the current status of the project, a review of the project minimum viable product

(MVP), user feedback from MVP tests and so forth. Figure 16 below illustrates feedback received during development.

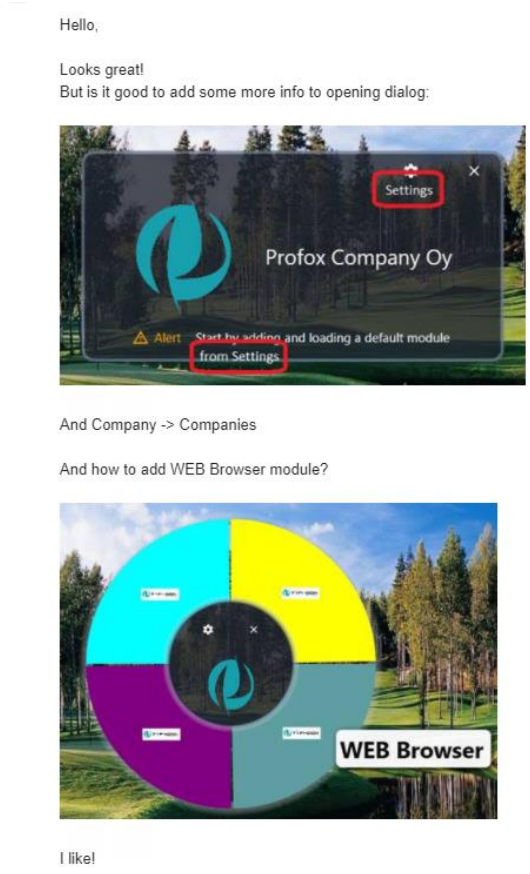

Figure 16. User feedback concerning the application.

Figure 16 above shows changes to the application that were proposed after user testing. These changes as well as numerous others were incorporated into the final UI application. The project was developed for about 3 months and the final version of the application can be viewed on the next page in Figure 17.

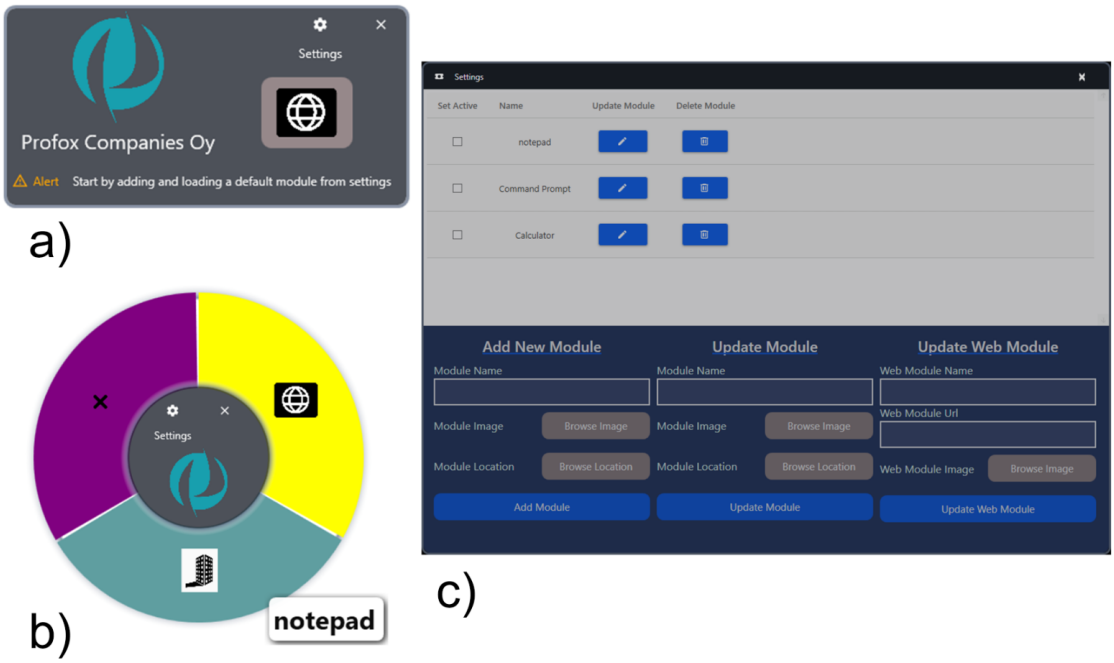

Figure 17. The final version of the application user interfaces. a) The default UI. b) The circular UI with additional modules. c) The settings UI.

Figure 17 above shows three different UIs. The default UI only has the Web module button. The circular UI can create segments dynamically based on activated modules. Moreover, it displays a tooltip with information of the segment under the mouse. The tooltip adds to the user experience when using the application. Finally, the settings UI shows a data grid with all modules from the database. In addition, the UI enables users to perform operations such as activate modules for display, create new modules, update modules and delete modules.

3.5 Testing

Two unit tests were carried out over the project's development process. The initial test was done to check the success of the Create, Read, Update, and Delete (CRUD) operations. A new Console project was utilized for the CRUD tests since the console applications execute much faster as compared to WPF projects. Figure 18 on the next page illustrates the source code for the initial database tests.

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        #region Module Test
        IDataService<ProfoxUI.Models.Module> moduleService = new GenericDataService<ProfoxUI.Models.Module>(new AppDBContextFactory());

        // Test Create new Module
        moduleService.Create(new ProfoxUI.Models.Module { Name = "Res", Description = "USA" }).Wait();

        //Read Module DB
        Console.WriteLine(moduleService.GetAll().Result.Count());
        Console.WriteLine(moduleService.GetAll().Result);

        //Get one Module DB
        Console.WriteLine(moduleService.Get(1).Result);

        //Update Module DB
        Console.WriteLine(moduleService.Update(2, new ProfoxUI.Models.Module() { Name = "Bug", Description = "Fixed" }).Result);

        //Delete Module DB
        Console.WriteLine(moduleService.Delete(3).Result);
        #endregion
```

Figure 18. The database unit tests.

As can be viewed from Figure 18 above, the unit tests were done on simple CRUD operations on the database. The operations involved creating a new module, reading only one and then all records from the database, updating a specific module's data and finally deleting the module. The success of the tests meant that similar CRUD operations could be performed on the active module objects.

The second unit test was carried out on the UI's ability to change based on changes in the state of the application. The steps for testing the UI were as follows: Three different UI layouts were created; the main layout had a navigation menu at the top of the layout; the navigation menu had three buttons each linked to its target layout; when either of the buttons was clicked, the main window loaded the required UI layout. The process of changing the layout was done through delegate commands since the navigation menu was an observable object. Therefore, if the status of the observable navigation menu changed then a property changed notification event was prompted. Figure 19 on the next page illustrates the code for the navigation menu observable object including the instantiation of the delegate command.

30 State/Navigators/Navigator.cs

```
@@ -0,0 +1,30 @@
+ using ProfoxUI.Commands;
+ using ProfoxUI.Models;
+ using System;
+ using System.Collections.Generic;
+ using System.ComponentModel;
+ using System.Text;
+ using System.Windows.Input;
+
+ namespace ProfoxUI.State.Navigators
+ {
+     public class Navigator : ObservableObject, INavigator
+     {
+         private TestVMBase _currentVM;
+         public TestVMBase CurrentVM
+         {
+             get
+             {
+                 return _currentVM;
+             }
+             set
+             {
+                 _currentVM = value;
+                 OnPropertyChanged(nameof(CurrentVM));
+             }
+         }
+
+         public ICommand UpdateCurrentVMCommand => new UpdateCurrentVMCommand(this);
+
+     }
+ }
```

Figure 19. The source code for the navigation menu.

Figure 19 above shows sample code utilized in the UI layouts unit tests. The code snippet has been sourced from a previous commit on Git. Moreover, it is possible to view how the layout is changed. The code on Figure 19 line 23 shows how the property changed notification sets the new ViewModel of a publicly accessible “CurrentVM” variable.

3.6 Publishing

According to the Microsoft documentation, applications can be published through two modes in .NET. The first mode involves publishing framework-dependent applications. The mode results in producing only the application and its dependencies. Thus, a user is required to install the .NET runtime separately. The second mode is self-contained publishing. An application that is self-contained means that includes the application and

its dependencies, and the .NET runtime and its libraries. Therefore, users only have to run the application without having to separately install the .NET runtime. The main advantages of utilizing the self-contained publishing is that it's possible to target a specific platform and the developer can control the .NET version deployed. [40.] The project was published using the self-contained mode. Figure 20 below illustrates the features of self-contained publishing.

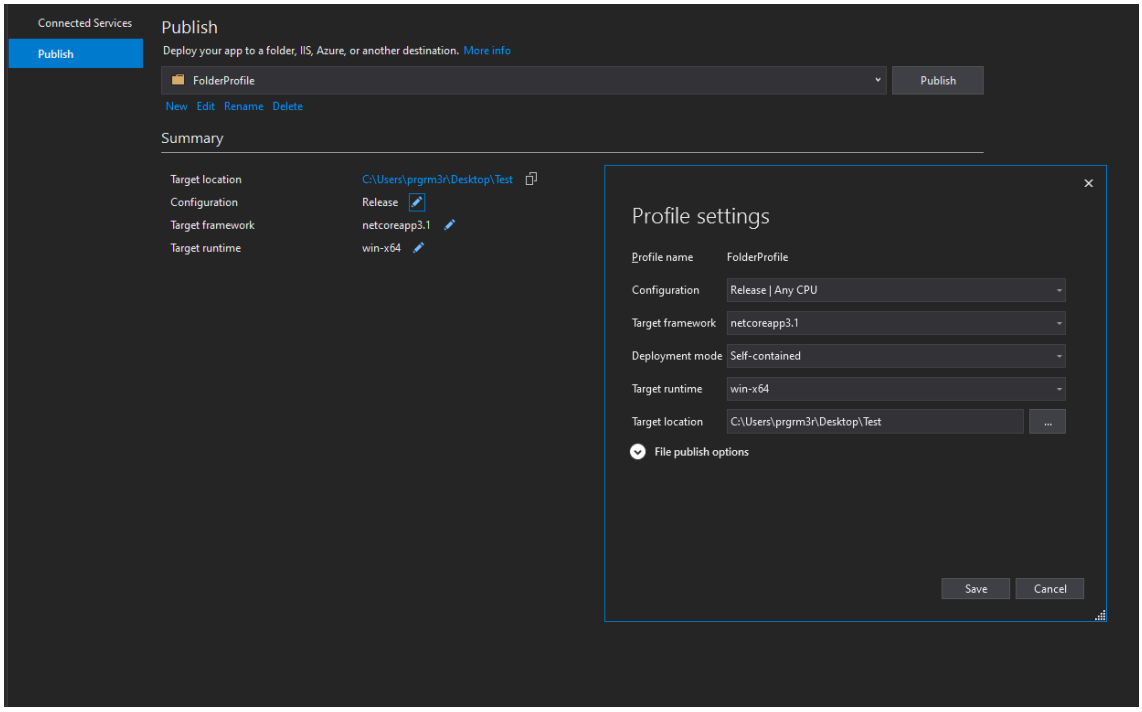

Figure 20. The selected options for a self-contained publish.

Figure 20 shows that the UI application was published as a self-contained application targeting the Windows x64-based processor. Therefore, it was possible for users to launch the application using the created executable file without having to perform additional installations.

4 Discussion

This chapter will review the results observed during the development of the UI application. The case company commissioned the project owing to the need to improve its customers' user experiences when interacting with software module products. To execute the project, a research and development approach was adopted. The research focused on integration methods for an OS software and an application software. Further, the research was utilized to inform design decisions such as the application architecture, the choice and purpose of components integrated and finally, a consideration of users' expectations, psychology and limitations. The development of the application focused on implementing concepts from the software requirement specification based on the research. The research and development approach is not unique to software development as the concept is utilized by various other industries such as in manufacturing, medical pharmaceuticals and so forth.

The result of the project was the creation of a Windows desktop UI application. The features of the application included the ability to launch multiple software modules from a single interface, the storage of module data to a database system, user notifications and a modern layout design. Moreover, the application was published as a self-contained application with a relatively large size of 165 MB. However, the benefits of such a deployment is that users can execute the application without having to install additional software. In addition, the application had the .NET runtime environment and its libraries included which meant the application was not dependent on the versions locally installed by the end users.

The main challenge during the project was the implementation of the doughnut UI layout. The layout is comprised of smaller arc segments that could be drawn either programmatically or through existing dependencies. The adopted solution was to import Microsoft expression drawing dependency for drawing and displaying the arc. Hence, it was programmatically possible to determine what software module was associated with each arc. The arc segment only displayed the software module icon prompting a need for a notification system of what each segment represented. Typically, a tooltip was the obvious solution and the size of the tooltip needed to dynamically change relative to the name of the software module.

As stated in the introduction, “The goal of the thesis project is to create a Windows desktop UI application that would launch modules from a centralized UI”, the results from the created UI application show that the target was achieved. Further, the implementation of the application was based on concepts highlighted during research. For instance, the layout was designed upon review of existing UI solutions, the integration of components was guided by the multi-layered multi-component based model and finally, the application incorporated features that improved user experiences when utilized such as notifications and animations.

The created application is reliable in performing the needed functions. Moreover, the application can launch other software programs also installed on the customer’s PC that are not the case company’s products. As a result, it is expected that customer experiences will be tremendously improved consequently boosting the financial benefits of the case company. However, a distinct drawback to the application is the lack of a feature that can enable the collection of usage data. The statistical analysis of usage data can yield important insights such as how the application is utilized, a need for possible performance improvements and the probable causes of functional errors or bug reports. The usage data collection feature is a potential feature candidate for software improvements in future application versions.

5 Conclusion

The goal of the thesis project was to create a Windows desktop UI application that would launch modules from a centralized UI. To achieve this goal, the application had an integrated database system that stored information about software modules. The project was developed for about 3 months and resulted in the successful implementation of a UI application.

During development, all the predefined software specifications were met and unit tests for the database and layout were performed. In addition, users' suggestions and feedback were incorporated as improvements to the application's features. The application was published as a self-contained executable with a size of 165 MB. The purpose of the thesis project was to improve the user experience of module software provided by the case company. An improved experience for customers makes the case company more competitive in its market segment.

The UI application was the first version created during the software development process. Subsequently, there were three beneficiaries of the project: The developer gained additional experience in creating production software; the case company created software that would improve its customers' experience; the customer interaction with software modules would be improved. In addition, future versions of the application can improve the software by incorporating more features such as personal customizations, a measure of usage metrics, support for module subscriptions or purchases and lastly, the incorporation of helpful resources such as learning material for the module software.

In conclusion, the thesis project succeeded in creating a UI application that could launch different software modules from a centralized UI. The application was targeted to the Windows OS desktop and had an executable that would launch the application without a need for additional software installations.

References

1 Novac, Ovidiu Constantin; Novac, Mihaela; Gordan, Cornelia; Berczes, Tamas & Bujdosó, Gyöngyi. 2017. Comparative study of Google Android, Apple iOS and Microsoft Windows Phone mobile operating systems. 2017 14th International Conference on Engineering of Modern Electric Systems (EMES), 2017, pp. 154-159. IEEE Digital Library. Accessed on 15 February 2021.

2 Ilyas, Muhammad; Khan, Siffat Ullah & Rashid, Nasir. 2020. Empirical Validation of Software Integration Practices in Global Software Development. SN Computer Science, 2020, Article 157, pp. 1-23. Springer Digital Library. Accessed on 15 February 2021.

3 Kempf, Torsten; Ascheid, Gerd & Leupers, Rainer. 2011. Multiprocessor Systems on Chip. New York: Springer. Electronic book. Springer Digital Library. Accessed on 16 February 2021.

4 Garg, Ruchi & Verma, Garima. 2017. Operating Systems: A Modern Approach. Bloomfield: Mercury Learning and Information. Electronic book. ProQuest EBook Central. Accessed on 17 February 2021.

5 Yao, Yanjun; Wan, Lipeng & Cao, Qing. 2014. System Architecture and Operating Systems. The Art of Wireless Sensor Networks, 2014, Volume 1, pp. 697-738. Springer Digital Library. Accessed on 17 February 2021.

6 Meng, Jian Liang & Li, Chao. 2013. Optimized Design of Employment Management System with WebService. Applied Mechanics and Materials, 2013, Volume 401-403, pp. 1931-1934. ProQuest Digital Library. Accessed on 21 February 2021.

7 Dayyani, Basel. 2016. Software architecture design and development of multi-layer highly modular platform using intelligent components for dynamic big data analytics. 2016 4th International Symposium on Computational and Business Intelligence (ISCBI), 2016, pp. 45-53. IEEE Digital Library. Accessed on 21 February 2021.

8 Haitao, Wang & Xing, Chen. 2011. Study of a Component Library Model Based on Four-Layer Architecture. 2011 4th International Conference on Intelligent Networks and Intelligent Systems, 2011, pp. 101-104. IEEE Digital Library. Accessed on 22 February 2021.

9 Solmz, Fritz. 2012. What is Software Architecture?. In Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference (SAICSIT '12), 2012, pp. 363-373. ACM Digital Library. Accessed on 22 February 2021.

10 Liu, Yajun. 2016. Analysis and Application of Interface Design Elements for Mobile Platform. 2016 International Conference on Smart City and Systems Engineering (ICSCSE), 2016, pp. 171-174. IEEE Digital Library. Accessed on 22 January 2021.

11 Laila, Siti Nur; Sabariah, Mira Kania & Suwawi, Dawam Dwi Jatmiko. 2016. UI Design of Collaborative Learning App for Final Assignment Subject Using Goal-Directed Design. 2016 4th International Conference on Information and Communication Technology (ICoICT), 2016, pp. 1-6. IEEE Digital Library. Accessed on 26 January 2021.

12 Hussmann, Heinrich; Meixner, Gerrit & Zuehlke, Detlef. 2011. Model-Driven Development of Advanced User Interfaces. Berlin, Heidelberg: Springer. Electronic book. Springer Digital Library. Accessed on 26 January 2021.

13 Jylhä, Henrietta & Hamari, Juho. 2020. Development of measurement instruments for visual qualities of graphical user interface elements (VISQUAL): a test in the context of mobile game icons. User Modeling and User-Adapted Interaction, 2020, pp. 949-982. Springer Digital Library. Accessed on 26 January 2021.

14 Rougier, Nicolas & Esfahbod, Behdad. 2018. Digital Typography: 25 Years of Text Rendering in Computer Graphics. In ACM SIGGRAPH 2018 Courses (SIGGRAPH '18), 2018, Article 12, pp. 1-29. ACM Digital Library. Accessed on 28 January 2021.

15 Wang, Junxiang; Yin, Jianwei; Deng, Shuiguang; Li, Ying; Pu, Calton; Tang, Yan & Luo, Zhiling. 2018. Evaluating User Satisfaction with Typography Designs via Mining Touch Interaction Data in Mobile Reading. In Proceedings of the 2018 CHI Conference on human Factors in Computing Systems (CHI '18), 2018, Paper 113, pp. 1-12. ACM Digital Library. Accessed on 28 January 2021.

16 Hidayat, Tonny & Sungkowo, Bayuarga Damar. 2020. Comparison of Memory Consumptive Against the Use of Various Image Formats for App Onboarding Animation Assets on Android with Lottie JSON. 2020 3rd International Conference on Computer and Informatics Engineering (IC2IE), 2020, pp. 376-381. IEEE Digital Library. Accessed on 28 January 2021.

17 Wang, Wallace. 2019. Pro iPhone Development with Swift 5. Berkeley, CA: Apress. Electronic book. Springer Digital Library. Accessed on 28 January 2021.

18 Best, Janet. 2017. Colour Design: Theories and Applications. Cambridge: Elsevier Science and Technology. Electronic book. ProQuest EBook Central. Accessed on 23 February 2021.

19 Rhyne, Theresa-Marie. 2017. Applying color theory to visualization. In ACM SIGGRAPH 2017 Courses (SIGGRAPH '17), 2017, Article 10, pp. 1-160. ACM Digital Library. Accessed on 23 February 2021.

20 Adinda, Prilly Putri & Suzianti, Amalia. 2018. Redesign of User Interface for E-Government Application Using Usability Testing Method. In Proceedings of the 4th International Conference on Communication and Information Processing (ICCIP '18), 2018, pp. 145-149. ACM Digital Library. Accessed on 23 February 2021.

21 Albert, William & Tullis, Thomas. 2013. Measuring the User Experience: Collecting, Analyzing and Presenting Usability Metrics. San Francisco: Elsevier Science and Technology. Electronic book. ProQuest EBook Central. Accessed on 23 February 2021.

22 McWherter, Jeff & Gowell, Scot. 2012. Professional Mobile Application Development. Somerset: John Wiley & Sons, Incorporated. Electronic book. ProQuest EBook Central. Accessed on 23 February 2021.

23 Dowden, Martine & Dowden, Michael. 2019. Why Should I Care About Accessibility?. Approachable Accessibility, 2019, pp. 1-18. Springer Digital Library. Accessed on 23 February 2021.

24 Kritikos, Kyriakos; Plexousakis, Dimitris & Paterno, Fabio. 2014. Task Model-Driven Realization of Interactive Application Functionality through Services. Association for Computing Machinery (ACM), 2014, Article 25, pp. 1-31. ACM Digital Library. Accessed on 23 February 2021.

25 Holzberg, Carol S. 2014. Adobe Creative Cloud. Tech & Learning, 2014, Volume 34 Issue 8, p. 22. ProQuest Digital Library. Accessed on 12 January 2021.

26 Anonymous. 2012. Press Release: Autodesk Adds Autodesk Inventor and Autodesk Revit Structure to Autodesk Plant Design Suite 2012. Dow Jones Institutional News, 2011. ProQuest Digital Library. Accessed on 19 January 2021.

27 Anonymous. 2016. AutoCAD 2017 launched with enhanced usability and design features. DataQuest, 2016. ProQuest Digital Library. Accessed on 19 January 2021.

28 Kirby, Lance; Krygiel, Eddy & Kim, Marcus. 2017. Mastering Autodesk Revit 2018. Indianapolis: John Wiley & Sons, Incorporated. Electronic book. ProQuest EBook Central. Accessed on 19 January 2021.

29 Wilson, Kevin. 2014. Using Office 365 with Windows 8. Berkeley: Apress. Electronic book. Springer Digital Library. Accessed on 19 January 2021.

30 Tan, Fei; Wei, Zhi; He, Jun; Wu, Xiang; Peng, Bo; Liu, Haoran & Yan, Zhenyu. 2018. A Blended Deep Learning Approach for Predicting User Intended Actions. IEEE International Conference on Data Mining (ICDM), 2018, pp. 487–496. IEEE Digital Library. Accessed on 12 January 2021.

31 Troelsen, Andrew & Japikse, Philip. 2017. The Philosophy of .NET Core. In: Pro C# 7, 2017, pp. 1245-1253. Springer Digital Library. Accessed on 24 February 2021.

32 Strauss Dirk. 2020. Getting Started with Visual Studio 2019. Berkeley: Apress. Electronic book. Springer Digital Library. Accessed on 24 February 2021.

33 Software Falafel. 2013. Designing in Blend. Pro Windows Phone App Development, 2013, pp. 449-516. Springer Digital Library. Accessed on 24 February 2021.

34 Nagel, Christian; Glynn, Jay & Skinner Morgan. 2014. Professional C# 5.0 and .NET 4.5.1. Somerset: John Wiley & Sons, Incorporated. Electronic book. ProQuest EBook Central. Accessed on 24 February 2021.

35 Joshi, Bipin. 2017. Introducing XML and the .NET Framework. Beginning XML with C# 7, 2017, pp. 1-28. Springer Digital Library. Accessed on 24 February 2021.

36 Bertino, Nic. 2012. Modern Version Control: Creating an Efficient Development Ecosystem. In Proceedings of the 40th annual ACM SIGUCCS conference on User services (SIGUCCS '12), 2012, pp. 219-222. ACM Digital Library. Accessed on 24 February 2021.

37 Mata-Toledo, Ramon & Monger, Morgan. 2011. Utilizing the ADO.NET entity framework in database courses. Journal of Computing Sciences in Colleges, 2011, Volume 26 Number 3, pp. 93-97. ACM Digital Library. Accessed on 25 February 2021.

38 Osman, Mohd Hafeez & Zaharin, Mohd Firdaus. 2018. Ambiguous software requirement specification detection: An automated approach. In Proceedings of the 5th International Workshop on Requirements Engineering and Testing (RET '18), 2018, pp. 33-40. ACM Digital Library. Accessed on 6 January 2021.

39 Rachel, Alt-Simmons. 2015. Agile by Design: An Implementation Guide to Analytic Lifecycle Management. Hoboken: John Wiley & Sons, Incorporated. Electronic book. ProQuest EBook Central. Accessed on 12 January 2021.

40 Microsoft Documentation. 2021. .NET application publishing overview. Online. 5 February 2021. Microsoft. < https://docs.microsoft.com/en-us/dotnet/core/deploying/ >. Accessed on 26 February 2021.