Student Name
Nicholas W. W. H. (XXX-XXXXXXXXX)

Project Supervisor
XXXXXXXXXX

September 2013

Investigation of Human-Computer Interaction using Algorithmic Application as a Solution for the Window Management Limitations Found in Desktop Environments with Windows 7 as an Example

---

Word Count:	12,928
Product & Development Files:	Uploaded as ZIP File

A dissertation submitted in partial fulfilment of the University of Greenwich's
BSc. (Hons) Computing (3+0) [SJ] Year 3

Abstract

This dissertation addresses the window management limitations found in Windows 7 in terms of human-computer interaction, usability and cognitive psychology. Cognitive analysis was performed using Keystroke-Level Models to study the hierarchy of user interactions, mainly in hand movements to and from the keyboard as well as hotkey executions with existing window management software. The project also uses user-centered design methods to analyze and determine the usability of different window arrangements on computer screens. A usability experiment was conducted to study the relationship between the complexity of screen layouts against task effectiveness, efficiency and user satisfaction. Adhering to usability standards and user-centered evaluation methods, the project goes through an iterative development process on a working application which maps real-life methods of managing documents into computer-based interactions aimed to significantly increase multitasking capabilities and efficiencies by reducing preparation time, number of task iterations per window, as well as completely removing point-and-click operations.

Preface

This dissertation is an original, unpublished work by the author, Nicholas W. W. H., submitted in partial fulfilment of the University of Greenwich's BSc. Computing programme. Published materials, framework or methodology used in the analysis stages of the project are attributed to Kieras (1993), Nielsen (1994) and Comber & Maltby (1995) while application development adapted programming concepts, interface elements and source code from De Smet (2006), Hartikainen (2007), Brooks (2009), Rutland (2010) and Dinham (2011). All requirements analysis, layout design, interface design, algorithm design, source code development and evaluation is done independently by the author using the combined methodology aforementioned with code adaptations from the programmers mentioned.

The idea of investigating human-computer interaction in terms of window management came from a standard day in the college in the first semester, third year of the degree programme. It was the class of this subject, XXXXXXXX where the Project Module Leader, XXXXXXXXX was busy arranging windows on his computer in order to compare 2 windows side-by-side, which he needs to process and write information on a third window at the same time. It took some time just to drag the windows to a desired position.

Upon examining this daily computer task, it caused a retrospection on my own daily tasks. I recall having the same trouble. Not only that, various window management limitations reminded me of the frustration of not being able to close all windows at once without logging off, setting a window to be always on top, fit all windows to screen, and so on. This thought leads to even more questioning of user interface and interaction design in common desktop operating systems which sparks the interest in the research for this project.

Acknowledgement

I would like to take this opportunity to thank my Project Supervisor and Link Tutor, XXXXXXXXX. Without her past teachings on system development frameworks, continuous guidance and strong support, completing this project to this standard would be near impossible.

Also, my deepest gratitude to XXXXXXXXX who has lectured for the XXXXXXXX subjects of the University of Greenwich Computing programme. Her dedication in her teaching has motivated me to scientifically question and analyze user-centered design methods to be used for this project. I would also like to thank her, along with other university students for their attentive participation in the usability experiment of this project, providing extremely helpful input for further analysis.

Last but not least, I would like to extend my gratitude to my family members, especially my parents who have always taught me to question everything. Their endless motivation, support, teachings, reassurance and love played a huge impact on my wellbeing and education today which enables me to fully carry out this project without any obstacles.

Table of Contents

1.0 Introduction

In the world today, almost all personal computers have a desktop environment which provide a graphical user interface consisting of windows, toolbars and icons which maps the real-life office environment to a virtual interface for users to effortlessly access and edit files. This interface is crucial for daily computer usage for both beginners and experts as it enhances human-computer interaction (HCI) which greatly reduces the time taken to perform tasks and increases usability to allow any user in the world to easily learn and make use of the computer. Thus, this project was started to address windows management limitations and issues in order to improve users' control over windows for efficient multitasking as well as to provide a solution for reducing clutter on the screen, especially for monitors with small resolutions.

This report presents the full documentation of the research regarding HCI issues in terms of limitations in windows management, hotkey functionality and screen layouts in desktop environments, and also the development process of a standalone windows management background application as a solution for the limitations in the Windows 7 desktop environment, Aero. The developed application uses Visual C# with the implementation of the Win32 API, more specifically user32.dll, which is a dynamic-link library for the creation and manipulation of elements in the Windows user interface.

The project is managed using Rapid Application Development methodology to gather the requirements dynamically while highly focused on evolutionary prototyping and testing due to high user involvement when creating algorithms for windows and screen layouts. Each project phase is timeboxed to a specific deadline and controlled using MoSCoW prioritization to ensure all deliverables are completed.

2.0 Literature Review

2.1 Rapid Application Development & MoSCoW Prioritization

Rapid Application Development (RAD) is a software development approach which uses minimal planning in favor of rapid prototyping. This methodology was chosen for the project due to its user-centered methods and speed (Kerr & Hunter, 1994) to overcome time constraints due to other responsibilities and risks faced in the degree programme of this project. RAD sets up the project in four phases: the requirements planning phase which determines project scope, constraints and system requirements, user design phase which develops diagrams and models discussed in a user-centered approach, construction phase for rigorous programming and testing, and finally the cutover phase for complete integration and documentation. A model of this can be seen at the end of the Project Proposal in *Appendix G*. The project report uses terms such as "iterative development" which refers to the 2nd and 3rd phases of the RAD methodology used.

RAD makes use of timeboxes to ensure all tasks are completed within time; otherwise they are dropped based on the Must, Should, Could, Won't (MoSCoW) priority. MoScoW quantifies tasks into four groups of priority as discussed by Hatton (2008), Tudor and Walter (2006) where tasks with "MUST" priority are decisive tasks for a project's completion, "SHOULD" for tasks which are nice to include in a project, "COULD" for less important tasks and "WON'T" for future implementations. Studies and comparisons made with other prioritization methods found that MoSCoW is extremely easy to use, consistent and takes less time to perform while providing high user confidence (Ma, 2009). These software development methods were used to ensure smooth management and progression of the project and may be referred to throughout this report.

2.2 Usability Standards & Evaluation Methods

One of the earliest frameworks for usability design and evaluation was analyzed by Shackel (1991) who used an operational approach by defining usability as the system's capability in human functional terms. The essence of the operational approach is that it explicitly defines usability through human-computer interaction more on system functionality and user satisfaction. The framework outlined by Shackel specifies 3 usability criteria – utility, usability and likeability which depends on its context of use, to measure usability. These criteria and methods were also adapted and further revised under the International Organization for Standardization (ISO).

For human-computer interaction and user interface design of all software, ISO helps determine the definition of usability for users all over the world regardless of culture. Adhering to these standards allow for the satisfaction of requirements of users all over the world, crucial for organizations aiming for international business. The concept of usability is defined of the ISO 9241 standard by effectiveness, efficiency, and satisfaction of the user. Part 11 gives the following definition of usability:

- Usability is measured by the extent to which the intended goals of use of the overall system are achieved (effectiveness).
- The resources that have to be expended to achieve the intended goals (efficiency).
- The extent to which the user finds the overall system acceptable (satisfaction).

Apart from standards, usability experts performed extensive research into quantifying subjective aspects of user interaction through studies of human cognitive psychology such as through the Goals, Operators, Methods and Selection (GOMS) analysis first developed by Card et al (1983). Goals are results which users desire to achieve. Operators are the list of tasks needed to achieve the Goal while Methods are a group of Operators used for a single Goal. Finally, Selection are conditional paths which determine which Method to use. Kieras (1993) further developed the GOMS method to introduce Keystroke-Level Model, breaking down user interactions into direct tasks applied to the keyboard and mouse designed for short tasks, suitable for this project. Along with these guidelines, Nielsen (1995) has defined 10 usability heuristics based on a factor analysis of 249 usability problems. These heuristics will be used for the usability evaluation of existing applications, as well as the developed application of this project.

2.3 Hotkeys as an Interaction Method

Hotkeys are keyboard shortcuts in which a keystroke (single-key) or a combination of keys (chorded) are pressed on the keyboard to perform a task or function that would otherwise be normally accessed or performed with the mouse. Hotkeys are used in place of mouse clicks because some frequently used controls can be hard to access on the screen due to factors such as size, distance, clutter and complexity of the task. For example, the window docking function in Windows 7 which allows users to automatically move a window to the left and resizing it to fit half of the screen, can be performed almost instantaneously with the key combination, "Windows Button + Left Arrow Key." This function is otherwise performed with the mouse by simply dragging the window to the edge of the screen and wait for the "snap" animation.

In terms of human cognition, a study on Apple's human interface (Tognazzini, 1989) has found that accessing controls with the mouse is faster than the keyboard because the act of recalling keyboard shortcuts correctly, due to the large amount of possible functions, require a higher order of cognitive skill than merely searching for a graphical widget with a mouse. Users who are new or still learning to apply the keyboard shortcuts actually spend more time in doing so, but subjectively reported that hotkeys were faster in their opinion. This perceived increase in efficiency suggests that the inclusion of hotkeys may increase user satisfaction in an application. Besides that, Tognazzini's study has also pointed out that users can better "remember disconnected data when they are the source for that data," so adding customizable hotkeys can help users remember them better. For expert users, a study of keyboard shortcuts using Microsoft Word (Lane, Napier, Peres, & Sandor, 2004) demonstrates hotkeys to be the most efficient method of interaction in terms of time. The study shows that one would have to issue at least 450 commands to save 15 minutes per day, which although is not much for an individual, it can be significant for an organization. The problem with hotkeys thus lies on its learning process and the willingness of the user to apply them. A study finds that non-keyboard shortcut users are most motivated to use hotkeys when there is someone to train them and least motivated if they simply knew that it would save time (Peres and Tamborello II et al., 2004).

As such, the project must look into the mouse as one of the main interactions for the developed system, with the inclusion of a separate control panel and easy instructions to help users learn the hotkeys and its functions. Implemented hotkeys must also be customizable and consistent with the existing hotkeys in Windows 7 without any conflicts.

2.4 Screen Layout Design & Complexity

A screen layout is the arrangement and presentation of windows, toolbars and other graphical elements on the screen. In a desktop environment, the screen displays windows in a given resolution space with a limited number of pixels. Due to the various sizes of monitors in the market, users all over the world have varying screen resolutions which will affect the amount of space available for display as well as the usability of the interface.

In order to measure the usability of a particular screen layout, Bonsiepe (1968) proposed a layout complexity metric based on the horizontal and vertical alignment of objects and their positional alignment, which is used to determine the relative order and disorder of objects. Thus, windows in a grid layout with consistent alignment and size are of minimum complexity, whereas unaligned and scattered windows with different sizes are of maximum complexity. In another research, the measurement of layout complexity was applied to alphanumeric displays on computer terminals and its results suggest that the lower the layout complexity, the higher the usability (Tullis, 1983). Another study finds that the optimal usability is found in the middle between the two extremes of layout complexity as users not only performed more efficiently and effectively with a layout of decent complexity, they are also more satisfied with it in terms of aesthetics. An explanation provided for this is that "a screen with minimal complexity is boring to look at and has difficulties in using size and position cues to indicate the function of objects. On the other hand, a screen with maximum complexity is also not desirable as it can be visually confusing and less productive to use" (Comber & Maltby, 1995).

The problem that needs to be addressed in the research area of this project is the type of layout to use when automatically rearranging windows. In terms of proportions, the Golden Ratio, also known as Golden Section, is a well-known mathematical relationship from ancient Greek, proposed to be aesthetically pleasing due to its frequent appearance in geometry and was explicitly used by some twentieth-century artists and architects such as Dali and Le Corbusier in their design models. However, some studies have suggested that the Golden Section has no aesthetic significance (Boselie 1997; Davis & Jahnke 1991). A more specific test on information retrieval with the computer using the Golden Section as a screen ratio resulted in poor task performance in terms of time taken, accuracy and aesthetic value; it was the least preferred screen ratio of all among the test participants as results indicate that "the best ratio is 28:72 for aesthetic value and 23:77 for performance" and recommends that the 23:77 ratio be used due to the lesser impact of aesthetic measures (van Schaik & Ling, 2003).

2.5 Windows API

An API is an application programming interface which provide a means of interaction among software components, usually through different platforms or hardware specifications. The Windows API specifically refers to a number of different platform implementations such as Win16, Win32 and Win64. Almost all Windows programs interact with the Windows API. The number prefixed to the back of the API name such as Win32, indicates its compatibility with 32-bit systems. The Win32 API is designed for modern computers which use 32-bit operating systems and therefore is the selected API for the project. Win32 API is used in Windows 7 to provide access to resources such as processes, file systems and devices, kernel operations, functionality for graphical output, user interface, graphical controls, access to the operating system shell and networking capabilities. Thus, in order to manipulate windows, the developed application requires a method of interaction with the Win32 API.

The Win32 API is implemented in the C programming language which is not object-oriented. It also comes in the form of dynamic-link libraries which enables the sharing of data and functionality. However, these libraries are in machine code and require an interface to interact with. Functions within the Win32 API lack available wrappers. Wrappers are a thin layer of code which translates the interface. Without it, the native code does not have a compatible interface for high-level programming languages to work with, and runtime interoperability cannot be achieved. Using Microsoft technologies ensures easier integration. Therefore, Microsoft Visual C# was selected for easy creation and manipulation with Windows Forms as well as the .NET Framework for easy access and integration with the Win32 API. The .NET Framework contains ready-made wrappers which exposes the Win32 API to be used with high-level object-oriented programming. Research shows that the implementation of .NET technologies for exposing the Windows API as managed code allows easy and rapid development of functional programs than in any other implementation (Benton, Kennedy and Russo, 2004) in the Windows operating system.

3.0 Window Management Limitations in Windows 7

Windows 7 offers various window management functionality for managing multiple windows efficiently on computer screens. Windows 7 introduces the Aero desktop environment with various technologies such as Aero Snap to automatically stretch windows to fill in screen space and Aero Peek to set all windows to transparent in order for users to look at the desktop. In order to identify the limitations of window management, the full capabilities of Windows 7 must be listed and examined one-by-one. All windows management functionality was compiled from the Microsoft Windows (2013) website and was tabulated below.

Hotkey	Function
Win + Up	Maximize window.
Win + Down	Restore or minimize window.
Alt + F4	Close window.
Win + Left	Dock to left half of the screen.
Win + Right	Dock to right half of the screen.
Win + Shift + Up	Stretch the window to fill the top and bottom of the screen.
Win + Shift + Left	Move to left monitor.
Win + Shift + Right	Move to right monitor.
Win + Home	Restore or minimize all windows except the currently active window.
Win + T	Cycle between pinned windows on the taskbar, starting from the first.
Win + Shift + T	Cycle pinned windows in the taskbar backwards.
Win + Space	Preview the desktop.
Win + D	Shows the desktop.
Win + M	Minimizes all windows.
Win + Shift + M	Restore all minimized windows.
Win + Tab	Flips through windows in a 3D interface.

Table 3.0: Hotkey Functionality for Windows Manipulation in Windows 7

Standard window operations such as minimizing and maximizing windows were assigned hotkeys as seen in *Table 3.0*, but not for moving or resizing of windows. Window minimize and restore operations were not only designed for a single window, but also for all windows on the screen. Yet, there is no close operation for all windows with the exception of logging off or shutting down the computer. Thus, these standard window operations which were omitted in Windows 7 have to be iterated and tediously executed for each window, whereas the minimize and restore operations can be carried out for all windows at once. The reason for this omission is incomprehensible.

Windows 7 also offers a docking system which allows users to expand windows to fill in the left or right half of the screen, maximizing the use of screen space. For the docking functionality accessed through hotkeys, it only docks a maximum of 2 windows on either side of the screen. Any other windows which are docked will overlap the previously docked window. It is peculiar because Windows 7 also offers automatic stacking, tiling and cascading functions to arrange windows on screen (Microsoft Windows, 2013). This can be done by right-clicking the taskbar and selecting the arrangement option. For example, selecting the stacking option when 4 windows are opened automatically arranges them into a grid, positioned at each corner of the screen and equally sized, making maximum use of space. However, this could not be achieved with hotkey docking because the windows do not automatically resize and arrange themselves when new windows come in. This lack of run-time automatic rearrangement makes window manipulation rigid in terms of defining customized layouts. If the hotkey docking functionality arranges and resizes windows as new windows come in, users can put in as many windows as desired in different sections of the screen without delay, which is one of the main focus of this project apart from the investigation of predefined screen layout complexity over usability aspects.

Besides that, Windows 7 does not seem to provide space management solutions. Windows are not provided controls to be hidden from the desktop to reduce clutter on the screen and in the taskbar. As users open more windows, the taskbar fills up with various icons to the point where it becomes difficult to switch between windows. A method must be developed to properly manage the number of windows in regards to computer screen space. Window manipulation in terms of space management is a subject that deserves deeper research and analysis, as this project would delve into. All in all, Windows 7 does provide some excellent solutions for window management with the use of its Aero desktop technology. However, the limitations aforementioned in this section must be addressed.

4.0 Review of Existing Applications

The limitations and lack of window manipulation controls in Windows 7 are not particularly unnoticed by its users which has led to the development of third-party window management tools and utilities for more functionality. Three latest applications on the market were chosen to be evaluated, which are Chameleon Window Manager, WindowSpace and Actual Window Manager in terms of **appearance** by comparing interface aesthetics and complexity, **usability** through heuristic tests, and **utility** using Keystroke-Level Model – Goals, Operators, Methods & Selection (KLM-GOMS or KLM in short) for a cognitive analysis based on the hierarchy of interactions and time taken in performing tasks. The KLM-GOMS approach was chosen due to its simplicity as it is designed to break down simple tasks which take no longer than 5 minutes. It also allows for a quick analysis without the need for usability experts (Kieras, 1993). The KLM is done for a simple task of arranging 3 different windows, 1 window at the left half of the screen with full screen height while the other 2 at the right half of the screen, equally divided in height. After the KLM analysis is done to acquire a time estimation, a trial run is performed with a stopwatch to record the actual time taken in order to reflect on the accuracy of the KLM estimation. A comparison will be made to analyze the effect of different levels, types and numbers of operations on task efficiency.

4.1 Chameleon Window Manager

Chameleon Window Manager is a window manager software by NeoSoft Tools (2013) which allows users to customize the behavior of all windows and add buttons with additional functions to the window title bars. It does not make use of hotkeys, but includes additional controls on every window. It can perform the following functions:

- Reposition and resize windows using a predefined layout or by drawing.
- Move window to next monitor.
- Set transparency of window.
- Minimize window to tray.
- Minimize window to caption (title bar).
- Save window state (size, position, transparency, etc.)
- Apply a rule on size, position, etc. to all windows or windows with a specific name.

4.1.1 Appearance

Its layout appears to be inconsistent by having its own custom grey frame, its controls docked at the various sides of the window *(see Figure 4.1.1a)* with a lack of proper spacing. Reaching controls require frequent travelling of the cursor from one side to the other with great distance as well as precision to actually select the control which costs time. There are lack of icons used to indicate the functions that it can perform, aside from window icons and a few controls. This further costs unnecessary time taken to read and understand the program whereas a simple button or visualization could have portrayed its meaning faster. Due to its lack of hotkey functionality, many buttons were added onto the title bar of each window, causing clutter *(see Figure 4.1.1b).*

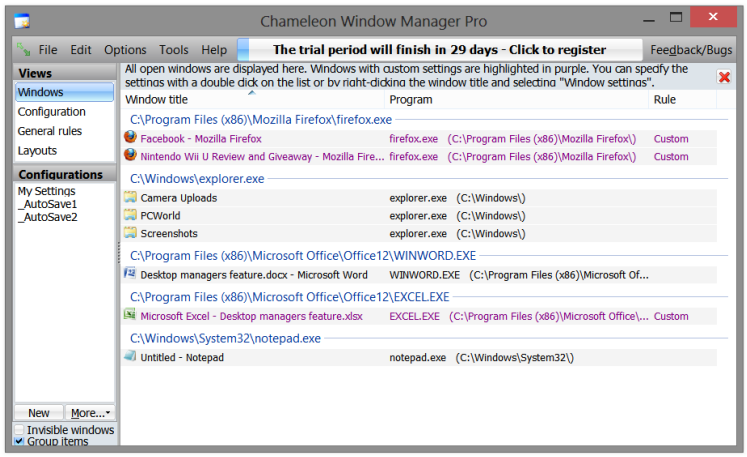


Figure 4.1.1a: Chameleon Window Manager Inconsistent Layout

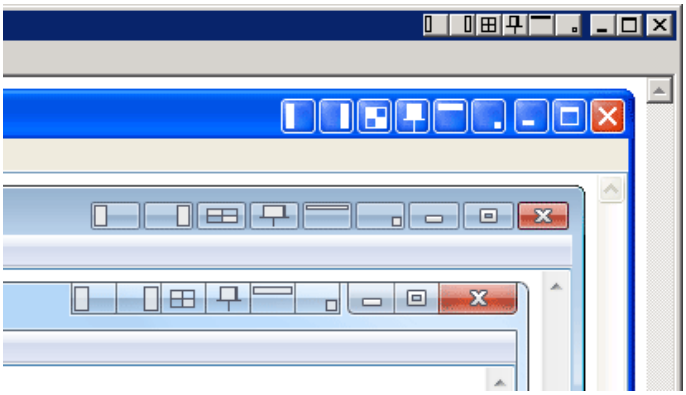


Figure 4.1.1b: Chameleon Window Manager Cluttered Title Bar

4.1.2 Usability

Chameleon Window Manager provides customization of all its features, including the adding and removing of buttons in each window, defining layouts as well as creating and loading configurations to be applied to each specific window. It is also highly compatible with many versions of Windows from Windows XP to Windows 8.1. A usability heuristics test was performed and can be seen in *Appendix A, Table A1* of this report, satisfying 4 out of 10 usability guidelines and found that Chameleon Window Manager makes use of technical terms when listing windows in its interface, confusingly portraying windows as processes with window classes and IDs without explanations for its technical information. Some functions do not work as expected; for example, some settings do not apply to certain windows and its cluttered controls on the title bar blocks controls from other windows which makes use of that space (an example of this can be seen later with Actual Window Manager). Although the user is able to define their own layouts, most of the configurability of other functions are minimal. Its usability problems mostly arise from its interface design, described as “cluttered and confusing” in a review article on PCWorld besides having inconsistent performance and bugs (Lancet, 2013).

4.1.3 Utility

According to the KLM analysis performed on Chameleon Window Manager *(see Appendix A, Table A2)*, it can be seen that its functionality is achieved through point-and-click interactions using only the mouse. The KLM estimated time to achieve the goal of arranging 1 window to the left half of the screen with full height and 2 windows to the right half with equally divided heights is 12.4 seconds, followed by a trial run achieving an actual time of 9.69s. Although it has been noted in the literature review that mouse interactions require a lower cognitive skill than remembering hotkey combinations, the Chameleon Window Manager uses small buttons and implements 2 layers of point-and-click navigation in order to select the desired layout for a window. Thus, the delay is mostly caused by the number of operations required to achieve the goal, twice as many compared to WindowSpace. During the trial run, these operations are also observed to require an amount of control and accuracy in using the mouse to select the desired items due to its small buttons. Due to these factors, mouse interaction is slower than hotkeys in this context. Users with poor mouse control are thought to perform poorly with the Chameleon Window Manager.

4.2 WindowSpace

WindowSpace is a lightweight desktop enhancement utility by NTWind Software (2013) for large monitors, widescreens and multi-monitor systems. It claims to provide window snapping features better than Aero Snap feature in Windows 7/8 and uses a variety of shortcuts including hotkeys and mouse key alternatives to existing buttons; for example, right-clicking on a window's Minimize button will send it to tray. It can perform the following functions:

- Move and resize windows with hotkeys.
- Snap windows to screen edges and other windows.
- Minimize window to tray.
- Hide a window.
- Automatic arrangement of windows, either by cascading or tiling.
- Close all windows at once.

4.2.1 Appearance

WindowSpace does not have its own interface except for a configuration window with checkbox and selection controls *(see Figure 4.2.1b).* It embeds its controls into windows and uses mainly hotkeys to perform its functionality. WindowSpace can auto-arrange windows using several pre-defined screen layouts such as the cascading layout and tiled layout *(see Figure 4.2.1a below)* with a limited number of windows to maintain readability. If the number of windows exceed the layout's expected number of windows, it will not arrange the remaining windows.

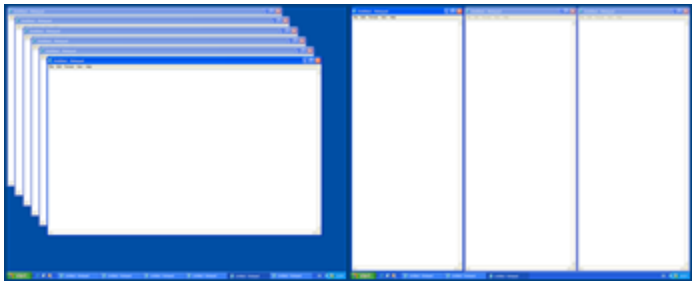


Figure 4.2.1a: Automatically Cascade or Tile Windows

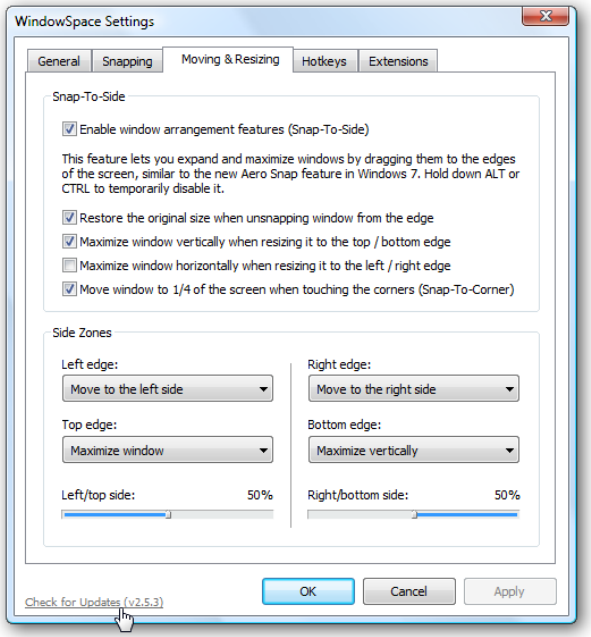


Figure 4.2.1b: Settings Window

4.2.2 Usability

WindowSpace provides a solution to most of the windows management limitations with highly customizable hotkeys and user preferences along with a simple and clean interface. Each feature can be turned on and off using checkboxes as the user see fit. With its straight-forward functionality, WindowSpace does its job extremely well with the pure use of hotkeys. It snaps onto windows and docks effectively. Nevertheless, some of the descriptions have technical terms which are not entirely clear to the average user and some terms are not even explained such as "MDI parent." The tile windows functionality only supports up to 3 windows. WindowSpace addresses window manipulation limitations only and serve to add those functions without much thought into the limitations of space. It is specifically created to suit large screens and does not automate or help in the arrangement or management of windows to increase space. Overall, WindowSpace is simple, efficient and very customizable for manipulating windows which almost passed the Nielsen's heuristics checklist *(see Appendix B, Table B1)* with 8 out of 10 of the guidelines followed.

4.2.3 Utility

The KLM analysis on WindowSpace *(see Appendix B, Table B2)* reveals that it has a lower number of operations required to achieve the goal compared to the others. Within these operations, the number of hand movement operations from keyboard to mouse and vice versa is double than that of other operations. This is caused by the distance between keys in the hotkey combination which requires both hands to be on the keyboard in order to execute. The main motivation in designing hotkeys which are harder to reach is to prevent users from performing an error-prone or irreversible task. Using hard-to-reach hotkeys by default for arranging windows – a task which is essentially the program's main purpose, causes unnecessary delay. Nevertheless, WindowSpace minimizes the amount of operations and selection methods which costs an estimated time of 7.8 seconds, and performed excellently at only 5.6 seconds in an actual trial run.

4.3 Actual Window Manager

Actual Window Manager was developed by Actual Tools (2013) to address limitations in window manipulation as well as desktop space through the use of virtual desktops and desktop profiles. It allows almost every possible action to be done with windows and offers plenty of user customization within those actions. However, it does not have automatic arrangement functions. The following are some of the functions it can perform:

- Configure each window individually in terms of startup position, priority, etc.
- Create and manipulate unlimited number of virtual desktops.
- Snap windows to screen edges and other windows.
- Minimize window to tray.
- Hide a window.
- Move and resize windows with hotkeys.
- Reposition and resize windows using a predefined layouts and sizes.
- Move window to next monitor.
- Set transparency of window.
- Minimize window to tray.
- Minimize window to caption (title bar).
- Save window state (size, position, transparency, etc.)

4.3.1 Appearance

Its interface is overwhelmed with information with descriptions on the functions of each component *(see Figure 4.3.1a).* The interface makes use of meaningful icons to describe functions and attributes of the system following consistent design patterns with labelled icons, proper spacing, as well as a top-to-bottom and left-to-right layout arrangement. It also embeds controls into windows without causing too much clutter as compared to the Chameleon Window Manager, but still causes an overlap with windows which make use of the window title bar space such as in the Google Chrome example in *Figure 4.3.1b.*

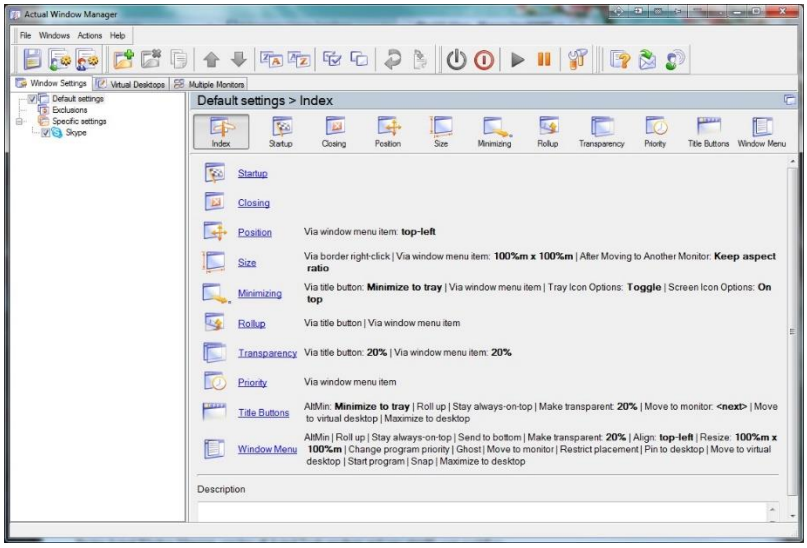


Figure 4.3.1a: Information Overload but with Clean and Consistent Layout

Figure 4.3.1b: Overlapping Controls

4.3.2 Usability

The Actual Window Manager has a large collection of functions which accurately addresses various limitations of window manipulation and desktop space. In the heuristics test *(see Appendix C, Table C1)*, its high compatibility with most version of the Windows operating system (from Windows XP to Windows 8.1) highly contributes to its usability and accessibility. All of the function hotkeys can be customized and new hotkeys can be added to perform functions based on user-defined values. However, the problem is with its interface and interaction. Window managers are supposed to improve usability in performing window tasks but this interface is hard to use and has a steep learning curve. Although the Actual Window Manager makes use of both hotkeys and mouse buttons to perform a huge variety of functions, they do not automate the tasks for the users. Users have to define sizes and positions for layouts or stick to the predefined values which do not automatically correspond to different screen sizes. Moreover, user-defined values for position and size are not assigned to hotkeys – meaning that any functions conforming to user-defined values must be manually configured or otherwise accessed with the mouse using a combination of right-clicks and left-clicks, costing unnecessary time. Some functions require 4-key combination hotkeys by default which are hard to remember and execute, yet reconfiguring them one-by-one also requires time and effort.

4.3.3 Utility

Based on the KLM analysis *(see Appendix C, Table C2)* for Actual Window Manager, the hierarchy of tasks in which the user has to perform is similar to WindowSpace but requires more steps to achieve the desired result. The reason is that in order to rearrange windows on the screen, a window has to be both repositioned and resized. These two tasks were combined in WindowSpace and Chameleon Window Manager, but not in Actual Window Manager. Therefore, an execution of two different hotkey combinations were required in order to fully rearrange a window. An added note is that the resize functions are fixed to a defined size; in this case, two types of resize operations are needed – one for the left side fully occupying the screen height and one for the equally divided heights of the right side. Cognitively, users also have to go through three mental preparations to recall and organize their actions to perform three different types of keystroke combinations. This slows down the user from achieving their goal and increases estimated time taken by 3 seconds to a total of 10.8 seconds compared to WindowSpace which similarly performs tasks using hotkeys. The trial run for Actual Window Manager was completed in 8 seconds time.

4.4 Key Issues to Use in Design & Implementation

	Chameleon Window Manager	WindowSpace	Actual Window Manager
Estimated Time (s)	12.40	7.80	10.80
Actual Time (s)	9.69	5.60	8.00
Heuristics Check	4 / 10	8 / 10	6 / 10
Number of Operations			
Mental Preparation	1	1	3
Hand Movement	1	6	6
Mouse Click	9	3	3
Pointing Mouse to Object	9	3	3
Key Press & Release	0	3	6
TOTAL:	20	16	21

Table 4.4: Comparison between 3 Existing Software

Table 4.4 above shows that WindowSpace performs the best in terms of appearance, usability and utility compared to the others. Its minimalistic design is easy to use and can be used as an example for this project. Also, it is most similar to the system developed in this project in terms of functionality compared to others due to its automatic arrangement of windows and its usage of hotkeys to address the windows management limitations in Windows 7. The only thing it lacks is more automatic functions and an actual interface. It also solely concentrates on using keyboard hotkeys as the only means of interaction which decreases flexibility for beginner users.

Comparing all three existing software in terms of task efficiency, WindowSpace performed the fastest due to its simpler keystroke model with hotkeys, followed by Actual Window Manager and Chameleon Window Manager. Based on a detailed view obtained from the usability heuristics check and KLM-GOMS analysis, it is seen that the hand movement from mouse to keyboard and vice versa, as well as the distance between hotkeys negatively impacts task efficiency. Thus, implementing default hotkeys which can be performed using only one hand is imperative for frequent tasks. Using mouse as the only means of interaction not only increases time taken in performing tasks due to the difficulty in acquiring small widgets on a large screen, but also takes away flexibility for expert users to perform faster. Overall, the system must implement a balance of easy-to-reach hotkeys for frequent tasks such as rearranging windows, hard-to-reach hotkeys for irreversible tasks such as closing all windows and lastly, an interface of buttons to support mouse interaction for beginner users.

5.0 Requirements Analysis

5.1 Questionnaire Survey

Due to the inconclusive studies and vague nature of user preferences, a questionnaire was created using SurveyMonkey to investigate user preferences on screen layouts at home and office environments as well as to determine the focus of the system based on the preferred methods in managing documents. The questionnaire was asked to home and office users situated in Malaysia and Singapore, with ages ranging from 16 to 42, some of which are laptop users. Question 1-3 are about the managing of documents in real-life, in which its responses can be mapped to computer interactions to initially determine the preferred method of organizing windows on screen, method of tracking important windows and position of controls. Question 4-10 are more specifically for computers to investigate the usage of hotkeys, windows and general preference in window properties. A total of 54 respondents participated in the survey and the results can be found in *Appendix D* of this report.

40.74% of the respondents prefer to stack their often accessed documents and put them aside as a method of organizing their desk in real-life, followed by 25.93% of respondents who prefer to store them somewhere else such as in a file or cabinet. As for keeping track of important documents, 64.81% of users prefer to store them in a labelled file, followed by 16.67% of users who prefer to label individual documents as important. This large difference in the preferred method of managing documents suggests that users tend to group up documents by category and hide their documents out of sight to avoid clutter on their desk. By mapping this behaviour into computer interactions, users should be allowed to group windows by category as a tracking method to bring up or hide away windows based on their determined importance by the user.

When asked of their preferred position of desk stationery, the most used position is top-right with 38.89% responses, followed by top-left with 29.63%, while the least used position is at the bottom and bottom-right with both having only 1.85% of the responses. As with most window controls, they are positioned at the top-right. Thus, the interface should be consistent and meet these expectations when positioning its controls as well.

Users tend to use hotkeys in their daily tasks. The questionnaire divides the usage of hotkeys into four ratings in which users can choose to determine how often they use them, with the rating of 1 being “Very Seldom,” and 4 being “Very Often.” 59.26% of the users respond with a rating of 3, and the average response of all users combined also resulted in the value of 3, which is “Often.” This determines the emphasis and configurability of hotkeys on the system based on usage.

59.26% of users find the drag-and-drop feature to be easy to use given the current window management features with comments such as “just a click away” and “user-friendly”, while the other 40.74% of users find it hard and tedious to use with comments such as “hotkeys are preferred,” “too many windows opened” and difficulty in dragging to the correct position. The fact that most users found it easy to drag-and-drop content suggests that the current windows alignment and snapping functionality works well enough that it does not need further improvement. Users who disagree however, commented for further support on the need for managing desktop space based on the number of opened windows as well as hotkey interactions.

Furthermore, users prefer to see visualizations of the original window when storing or manipulating simplified objects which represent the window with a majority percentage of 38.89%. Simplified objects are used to save space and allow for greater manipulation and presentation within the system interface. However, this majority percentage for window snapshots is closely rivaled by floating bubble icons with the support of 35.19% of the users, followed by 22.22% of users for text labels. Although the highest priority will be put into generating window snapshots, the significant amount of support by other users also indicate that a combination of icons and text would be a suitable approach to address almost all of the users’ needs. These icon and text attributes are considered to be added into the system at a later stage with a lower priority, and is added into the MoSCoW prioritization list.

Lastly, the general focus of the system is determined as majority of the users find window animation to be important (68.52%), work with a maximum of 6 to 10 windows on the screen at once (50%), prefer landscape windows over portrait or square (75.93%), and most comfortable working with a window size which covers at least half of the screen (24.07%). Based on all of the collected statistics, *Table 5.1* on the next page lists the attributes and some of the non-functional requirements in which the system should have.

Position of Controls	Top-Right of Screen
Hotkey Functionality	High
Method of Organizing Windows	Group by Category
Method of Determining Important Windows	Group by Category
Improvement of Window Snapping Functionality	Not Necessary
Importance of Animation	High
Number of Windows on Screen	6 – 10
Window Orientation	Landscape
Minimum Size of Main Window	50%
Simplified Windows Form	Window snapshots.

Table 5.1: Attributes & Non-Functional Requirements of the System

Based on the information incurred from the analysis, the minimum window size that users prefer working with covers 50% of the screen. In order to account for readability, the number of windows must be limited. Hypothetically, given the remaining screen width = 1, ratio sizes of 23/77 can only fit in 3 more windows, giving a total limit of 4 windows. Due to this problem, a proposed solution would be to have a control value, known as “Layout Limit” which groups windows into a separate category if the number of windows exceed the limit of the screen layout. The algorithm will automatically create new categories if the current category has exceeded its limit. By grouping windows together, users can add or remove windows from groups in addition to showing or hiding all windows from a specific group. This strategy is in coherence with user preferred methods in managing documents.

5.2 High Level Requirements with MoSCoW Prioritization

The tasks in *Table 5.2* below are for the high level requirements of the system prioritized based on the MoSCoW method. There are only two development iterations in this entire project. All tasks under the "MUST" priority are scheduled to be completed within the first iteration whereas tasks under the "SHOULD" priority are scheduled to be completed by the second iteration. Any extra time remaining is spent on tasks with "COULD" priority. Tasks with "WON'T" priority are taken into consideration for current tasks to prepare the system for its future implementation.

Task	Priority	Comment
Toggle automatic window docking	MUST	Required for primary function.
Dock windows on screen	MUST	Primary function for multitasking efficiency.
Rearrange all windows	MUST	Primary function to allow multitasking.
Add window into group	MUST	Grouping windows solves limitations.
Rearrange windows of a group	MUST	Enables multitasking by group.
Hide windows of group	MUST	Solves the screen space problem.
Show windows of group	MUST	Solves the screen space problem.
Show overlay of windows and controls	MUST	Provides an interface for beginner users to interact with the program.
List windows by group	MUST	Required for user to interact with groups.
Delete window group	MUST	Required to remove unwanted groups.
Make active window the main window of the group	MUST	Significantly improves multitasking.
Swap active window position with main window	SHOULD	Improves window manipulation in terms of multitasking capabilities.
Move active window	SHOULD	Improves window manipulation.
Resize active window	SHOULD	Improves window manipulation.
Close windows by group	SHOULD	Increases window manipulation efficiency.
Close all windows	SHOULD	Increases window manipulation efficiency.
Kill windows by group	SHOULD	Increases window manipulation efficiency.
Kill all windows	SHOULD	Increases window manipulation efficiency.
Rename window group	COULD	For user customization purpose only.
Start application at startup	COULD	Not important, for user convenience only.
Reconfigure hotkeys	COULD	For user customization purpose only.
Animation of window movements	WON'T	Does not benefit users or program except for user satisfaction. Implement in future.
Extra functions for compatibility	WON'T	Current project scope is for Windows 7 only.

Table 5.2: Task Priority List

Based on the initial Rapid Application Development Model with Timebox Plan from the Project Proposal *(see Appendix G),* the scheduled date for the first development iteration is from 10th of November 2013 to 10th of January 2014 which is a period of 2 months. Within these two months, tasks with the "MUST" priority must be completed, otherwise they are carried over to the second development iteration which is from 11th of January 2014 to 10th of March 2014, another period of two months. The "SHOULD" tasks however, are supposed to be dropped if they are unable to be completed within the second iteration. This ensures all top priority tasks are completed in order to satisfy the primary requirements and scope of this project.

5.3 Minimum System Requirements

The literature review and analysis helped determine the technologies to be used in the system. These technologies require that users have the following software and hardware components for the system to work, assuming all Windows 7's minimum requirements are already met:

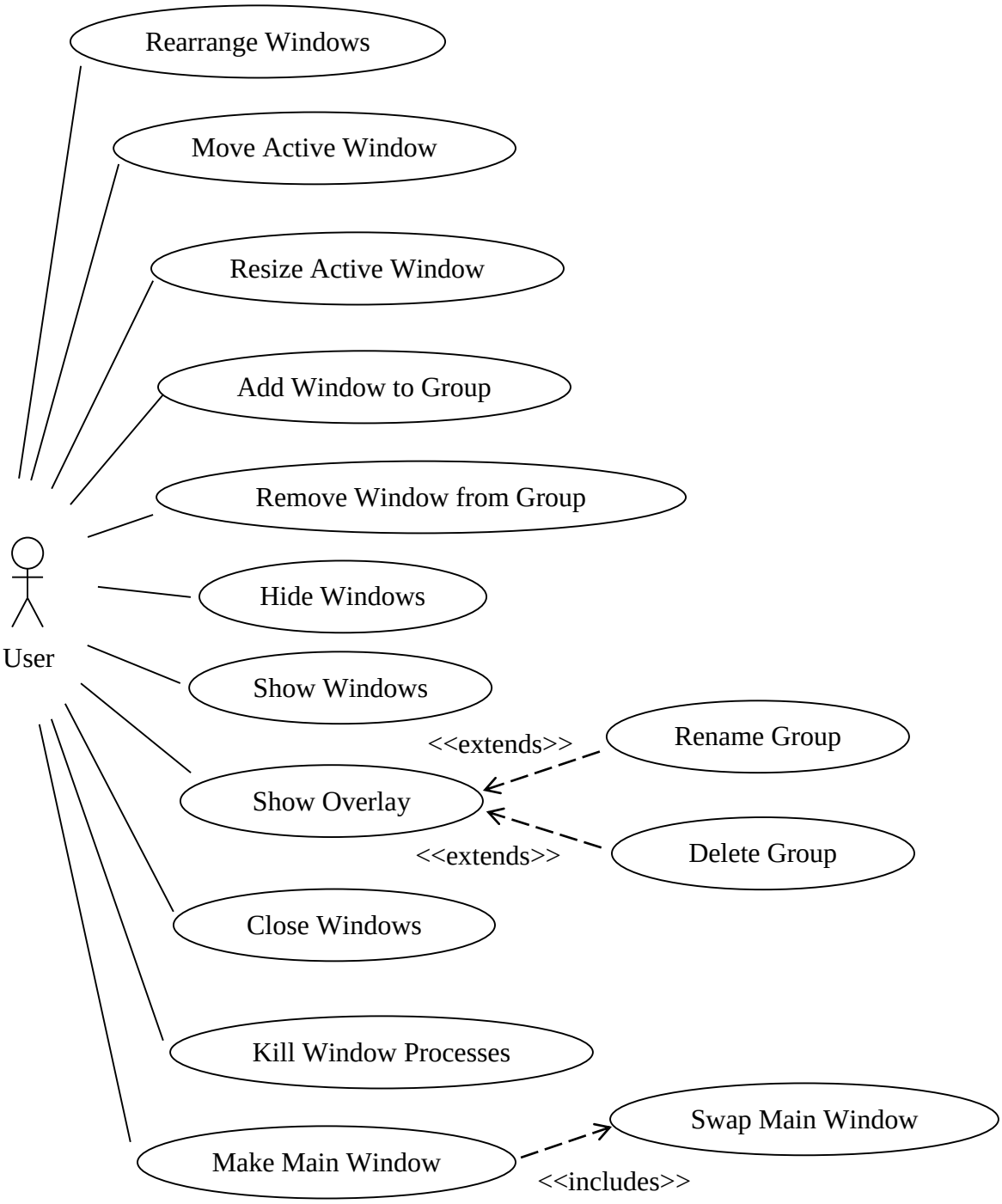
5.3.1 Software Requirements

- .NET Framework 4.0 or above
- Windows 7 (32-bit)

5.3.2 Hardware Requirements

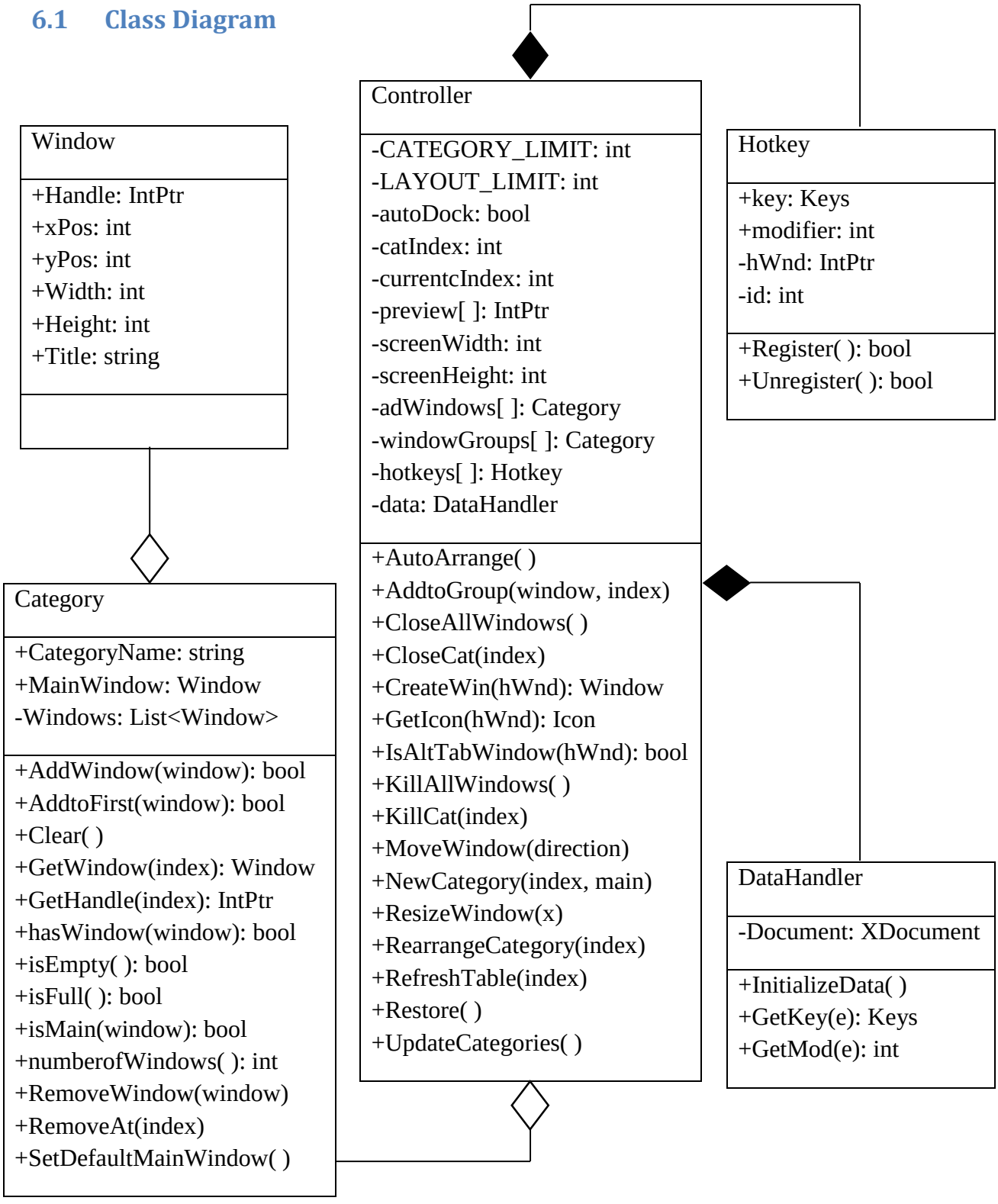
- 500KB HDD Space

5.4 Use Case Diagram



6.0 Design

6.1 Class Diagram



6.2 Interface Design

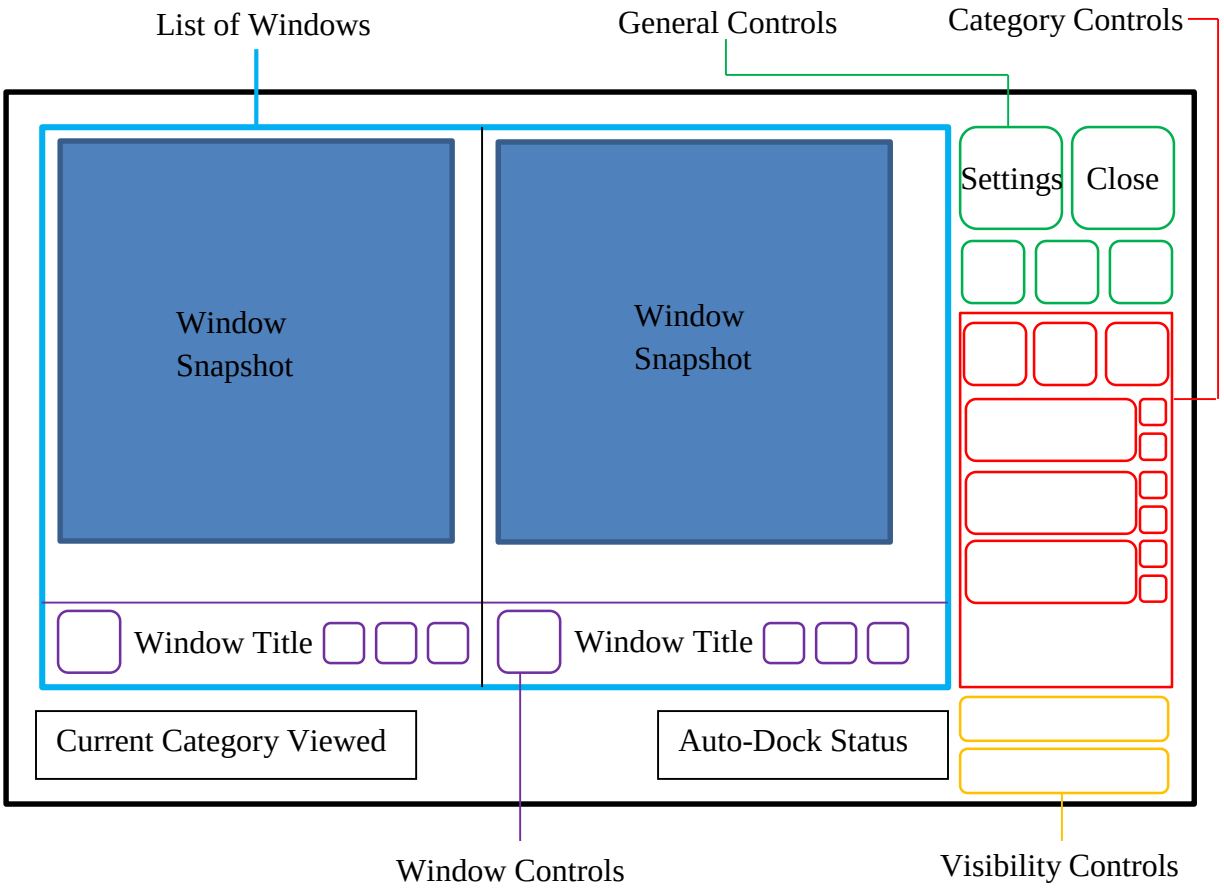


Figure 6.2: Interface Design

The interface design in *Figure 6.2* follows minimalistic concepts and is based on a grid layout following symmetrical approach using grids as discussed by Ngo and Ch'ng (2001). Composing and designing spaces with grids allow controls to be propagated consistently, allowing a more structured flow for users to access controls and interact with the application. The positioning of main controls on the top-right of the interface tailors towards user expectations in control positions, reflecting on the information obtained from the questionnaire.

6.3 Screen Layout Designs

6.3.1 The 50% + 23:77 Layout

This layout is drawn directly from the results obtained from the analysis, which is to have a main working window covering 50% of the screen, while the rest of the windows are tiled from top to bottom in a 23:77 ratio, with the last window taking up the remaining space *(see Figure 6.3.1).* Although the 23:77 ratio was applied indirectly to individual windows, the original experiment was based on the effectiveness of performing computer-based tasks on the screen. The screen displays content in a similar way; windows just divides the screen into small sections – smaller screens. Thus, the application of the ratio on individual windows may create the same effect of displaying content for high user performance due to its similarity.

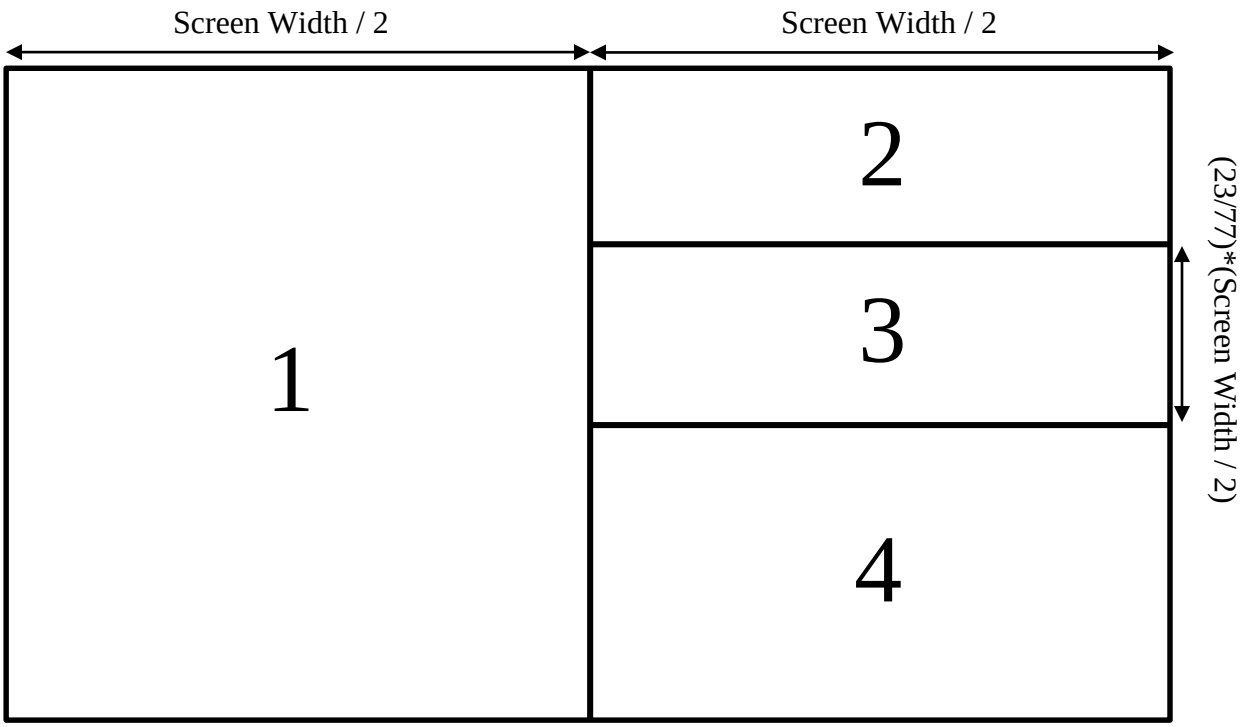


Figure 6.3.1: 50% + 23:77 Layout

6.3.2 Conformed Golden Spiral Layout

The Golden Spiral is a logarithmic spiral where the spiral gets further from its origin by a factor of the Golden Ratio for every quarter turn *(see Figure 6.3.2).* The Golden Spiral cannot be applied correctly to monitor screens because no monitor aspect ratio uses the Golden Ratio. Nevertheless, this layout was created to explore the flow of a similar arrangement based on the Golden Section using the Golden Ratio for the size and arrangement of windows. The Golden Ratio is expressed as φ which holds the value:

$$\varphi = \frac{1 \pm \sqrt{5}}{2}$$

Due to the aforementioned fact that computer screen ratios do not match the Golden Ratio as well as screen resolutions being different for each user, it is impossible to apply the spiral on computer screens. However, it is possible to apply the ratio to each individual window on screen, but layouts are based on the overall screen's width and height; applying it to individual windows does not accurately reflect the Golden Section theory as a screen layout. Because of this, the spiral will origin from the lower-left corner of the screen and conform to the screen's width and height while still partially using the golden ratio in determining measurements of a window.

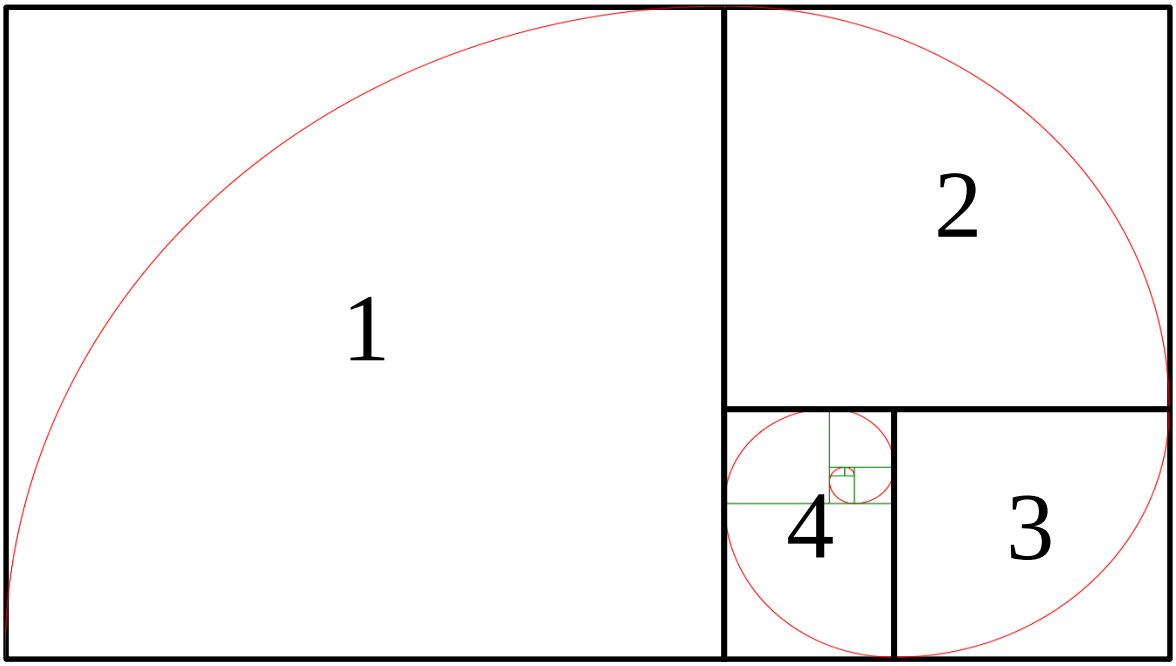


Figure 6.3.2: Conformed Golden Spiral Layout

6.3.3 Grid Layout

The grid layout is a test for minimum layout complexity based on Bonsiepe's concept (1968) as well as Comber and Maltby's evaluation method (1995) as mentioned in the literature review. A layout of 4 windows with minimum complexity would all have the same widths and heights aligned as symmetrical as possible following the Gestalt principles of visual balance where a psychological sense of equilibrium is achieved when visual 'weight' is placed evenly on each side of an axis (Chang, Dooley & Tuovinen, 2002).

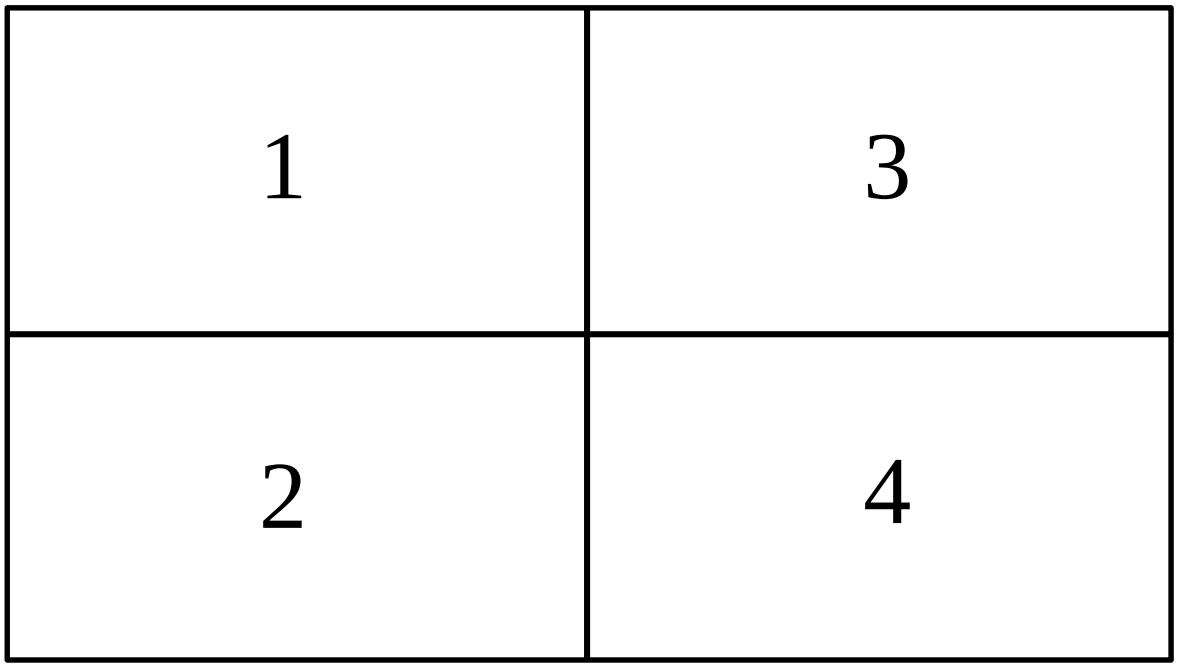


Figure 6.3.3: Grid Layout

6.3.4 Unaligned Layout

This layout was designed based on the same theory used in the grid layout but with high layout complexity. Windows are not aligned with one another at any side and have irregular spaces between them. Compared to the other layouts, this does not maximize the use of space which serves as a representation of daily computer screen states where windows are not aligned and are simply scattered throughout the screen.

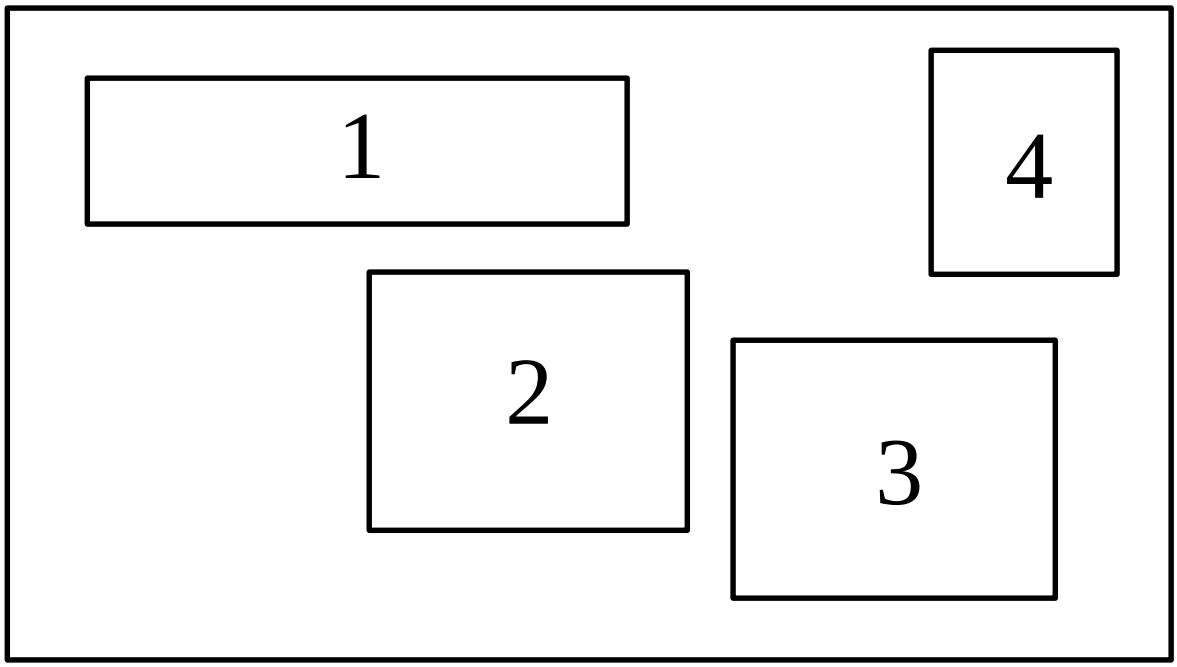


Figure 6.3.4: Unaligned Layout

6.4 Algorithm Design

Throughout the iterative development process of this project, screen layouts were tested and evaluated by manually setting window position and sizes in the application using fixed values, before an algorithm is designed. It is after the evaluation process which determines the accepted screen layout to be used for the application. Therefore, only the 50% + 23:77 layout which was accepted and finalized has a working algorithm which is documented in this section.

An algorithm is a fixed procedure of calculations designed to solve a problem based on known variables; in this case, the number of windows. The algorithm needs to rearrange windows in a set of position and sizes in a way that maximizes the use of screen space. As the determined method for handling large number of windows is by categorization, the algorithm must reset itself when the layout limit is reached – that is, when the designed screen layout can no longer support any more usable windows. Meanwhile, the algorithm must also set a main window for each category with its own size and position regardless of any other factors that may affect its arrangement. Therefore, it must find and arrange the main window, determine the remaining space available and then arrange the rest of the windows within the remaining space while checking the layout limit and ignoring the main window. This requires logical comparisons and incremental loops to manipulate each window. A pseudocode was written in *Figure 6.4* below.

```
// Constants
Widths of all windows are half of screen width, x.
Number of windows, n.

// Manipulative Variable
Y-position of current window, y.

Loop until all windows are arranged.
{
        Get current window in the loop.
        IF current window is main window,
        THEN
                Set its x-position = 0, y-position = 0, width = x, height = screenHeight

        ELSE IF twice the current y-position LESS THAN screen height minus current y-position,
        THEN
               Set its x-position = x, y-position = y, width = x, height = screenHeight - y

        ELSE
                newY = (23/77) * x
                ROUND UP newY
                Set its x-position = x, y-position = y, width = x, height = newY,
                ADD newY to y
}
```

Figure 6.4: 50% + 23:77 Algorithm Pseudocode

7.0 Development

The application was developed with Visual C# using Microsoft Visual Studio 2013, making use of .NET Framework 4.5.1 to take advantage of interoperability using Platform Invocation Services (P/Invoke) to integrate with machine language via dynamic-link libraries (DLL) of the Windows API. This ensures interoperability between high-level programming language with the native code, allowing direct manipulation over the Windows 7 user interface through DLL imports of the user32.dll library for user interface access.

An interface was developed based to ensure most of the controls can be accessed with the mouse for new users or if the user does not prefer using hotkeys to enhance usability. The icons used were from the Token icon set designed by a deviantART user named "brsev" which is free for personal use under the Creative Commons Attribution-Non-Commerical 3.0 license (Brooks, 2009). Credits to the Photoshop .PSD file provided which allowed other original icons used for this project to be developed using the same colour scheme and style, mainly the arrange by category, make active window, kill window process and minimize interface icon *(see Appendix H, Figure H1 for interface screenshot).* The interface allows users to directly manage window groups and offers user customization in renaming categories *(see Appendix H, Figure H4).*

Besides that, showing window snapshots within the interface is done by integrating with the Desktop Window Manager (DWM) API which collects full-sized bitmap images of all active windows at any point in time. The snapshots are not just Windows Forms objects which can be manipulated normally. Displaying window snapshots works by registering other window handles to the application's window handle, creating a reference from the memory spaces of other window processes to the application's process. The application stores these window handles which can then be manipulated using the DwmUpdateThumbnailProperties function call to set the size and position in which the window snapshots will display in the application's interface *(see Appendix H, Figure H3).* Some of the implementations were borrowed and modified from a tutorial on the DWM API for Windows Vista (De Smet, 2006). Obtaining window icons work in a similar method but only requires a call to the Win32 API to fetch the icon reference which is then converted into an icon object. Due to some problems in fetching icons from certain windows, Hartikainen (2007) proposed a solution by listing all possible ways in C# source code for obtaining different types of icons from windows. This code was adapted and used for the application.

Based on the usability experiment, the 50% + 23:77 layout was accepted to be further developed in the iterative development process. However, before the application is able to implement the algorithm for the layout, the application needs to select all "real" windows. This is because Windows 7 has plenty of windows hidden and running in the background which needs to be omitted. Using the EnumWindows function from the Win32 API to go through all windows in the system, each window must pass through a check to determine if they are "real" windows before any further manipulation is to be done. Some windows are also "Tool Windows" which are floating toolbars linked to a parent window; therefore, they are not individual windows that should be arranged on screen. Basically, windows which appear in the Windows Task Switcher brought up using the Alt + Tab key are "real" windows. In a Microsoft Developer Network blog article, the rule for determining windows which appear in the Alt + Tab list is the "most meaningful representative window from each cluster of windows related by ownership" (Chen, 2010). A post from Stack Overflow Q&A forums addresses this method, providing it in the form of C++ source code and further expands it by excluding windows with invisible title bars as well as tool windows (Dinham, 2011). This IsAltTabWindow function written in C++ was adapted and slightly modified for this project into the C# language with P/Invoke to import interoperable functions and data structures from the Win32 API.

For the rearrangement of windows, the application applies the 50% + 23:77 layout algorithm in a loop which checks through the list of windows for "real" windows. Once that is done, it checks if the current category is full and creates a new one if it is, automatically assigning the current window as the main window. This is because the user can add windows to categories, swap windows and rearrange them at any time, so a fixed counter is not a viable method. Since the main window of a category can be at any position in the window list at any given time, the loop also checks each window to see if it is the main window of the category, then immediately uses the SetWindowPos function from the Win32 API to arrange it in the fixed position and size on the left side of the screen based on the layout algorithm. Every other window which is not the main window is arranged from top to bottom at the right side (*see Appendix H, Figure H2)*. Eventually the current category will be full and the loop moves on to the next category. Other functionality is also achieved with other Win32 API functions such as using the SendMessage function to send a WM_CLOSE message which attempts to close windows *(Appendix H, Figure H5)* and GetProcessById to get and terminate window processes.

One important development progress in the application is the automatic switching of windows. The application implements an automatic docking function which automatically docks windows in the direction of pressed arrow keys, automatically resizing any window to ensure all windows fit into the docked space without overlapping each other *(Appendix H, Figure H6)*. As a result of the KLM cognitive analysis on existing software, it was seen that all 3 existing software did not automate the window switching task for the user. Users had to manually point and click each desired window to repeat the operation. Therefore, the application aims to automatically switch to another window, but it not as simple as it seems. In order for window switching to be effective, it must not switch to a previous window which the user has already manipulated. There were two approaches that were studied at this point:

1. Store each window reference that the user has manipulated. Then, enumerate all windows and check if the window has already been manipulated. If yes, ignore and go to the next window. If all windows were manipulated, clear the stored references.

2. Identify the back-most window and switch to that window.

Solution #2 was chosen because it was more consistent. Solution #1 would switch to a random window every time and take up memory space. Although Solution #1 is faster because it is more direct, Solution #2 benefits may benefit many operations in future by identifying window Z-Order, which is the ordering used in overlapping windows. A higher Z-Order indicates that the window is drawn on top of other windows and vice versa for a lower Z-Order. However, unlike other functionality, no tutorials, help or proposed solutions were found on the Internet to find and obtain the reference to the back-most window. This is because there are many unwanted invisible windows running in Windows 7 and there are no API functions to precisely perform this functionality. Through persistent studying and analysis of programming concepts, a solution was developed. First, the application loops through the Z-Order upwards from the current window, counting all higher order windows if they are valid windows. Once the loop has reached the top of the Z-Order, it counts how many valid windows the loop went through minus the current window. If the value is incorrect, then the loop starts over by moving on to the next window down the Z-Order. This operation repeats until there the counter and the number of valid windows are correct. This would mean that all other valid windows are on top of the current window. No more valid windows would be below the current window because all valid windows were already accounted for in the loop.

In order to make it a background application, Windows Forms provides functions to completely hide and show windows. Windows Forms also provide a control component called NotifyIcon which serves as a tray icon in the taskbar for notifications. By simply hiding the form, the NotifyIcon can be made visible in the tray for the user to restore at any point in time. The icon also has event functions which can perform tasks when clicked, such as to show a context menu to allow users to access functionality without restoring the main form. The next challenge from this point was to make it a startup application which automatically runs when Windows starts. This was achieved by simply adding a registry key containing the application path into Windows' startup path.

Lastly, for persistent and reconfigurable hotkey data to be remembered by the application, hotkey definitions are generated and stored in an XML (Extensible Markup Language) file because XML is a core technology used in the .NET Framework. It is fully supported as almost all parts of the .NET Framework use XML as the native data representation format. XML is also platform-neutral, making it one of the best technologies for future interoperability between different systems, not just Windows. Furthermore, by making use of Linq to XML technology provided since .NET Framework 4.0, adding, editing and retrieving data from file is easy and extremely efficient. Data operations are handled by the DataHandler class of the application. Then, the application has a Hotkey class to register it globally in the Windows operating system also by using functions imported from the Win32 API, namely RegisterHotKey and UnregisterHotKey based on a tutorial provided by Rutland (2010) in the Dream.In.Code forums.

8.0 Evaluation

8.1 Usability Experiment

An experiment was conducted with 6 home and office users who have been using personal computers for their daily tasks for at least 3 years, and worked with the Windows 7 operating system for at least 1 year. This amount of time spent in using computers ensures that they are familiar with mouse and keyboard interactions, as well as the Windows 7 interface. The aim of this experiment is to study usability on aspects defined in ISO 9241 which are effectiveness, efficiency and satisfaction based on screen layout complexity.

8.1.1 Test System Specification

System Unit:	Acer Veriton M2610G
Processor:	Intel Core i5-3330 CPU @ 3.00Ghz (4 cores)
Memory:	4096 MB
Mouse:	Acer SM-9020B Optical Mouse
Keyboard:	Acer PR-1101U (QWERTY US Layout)
Monitor:	Acer V196HQL (18.5”)
Resolution:	1366x768 (60Hz)
Aspect Ratio:	16:9
Operating System:	Windows 7 Professional 6.1, Build 7600 (32-bit)

8.1.2 Procedure

The test was done using a desktop computer in the computer labs located in SEGi College Subang Jaya. The computers are setup by having 4 windows opened, which are Google Chrome, Microsoft Word, Notepad and Windows Explorer. The Google Chrome web browser is set to display a page with a generated paragraph of text by a Lipsum Generator (2014) while Windows Explorer was navigated to the user folder *(C:\Users\user).* Each window is then added into a category one-by-one, to preserve their order when being rearranged by the program. This ensures each participant experiences the exact same arrangement of windows for each screen layout.

Before the test began, participants were told about the application and its functionality. They were given only one trial run to familiarize themselves with the arrangement and tasks they were about to perform. Then, each participant was asked to perform the following tasks:

1. Press the Shift+1 key to rearrange windows.
2. Drag to select the first Lorem Ipsum paragraph in Google Chrome.
3. Drag and drop the paragraph to Microsoft Word.
4. Type ‘done’ in the Notepad.
5. Double-click the My Documents folder in the Windows Explorer.

The participants were also told to ignore selection and typing errors and carry on as they normally would as the errors are recorded in the test. Four layouts were presented to the users in the following order: 50% + 23:77 Layout (FTL), Conformed Golden Spiral Layout (CGSL), Grid Layout (GL) and Unaligned Layout (UL). These tasks were repeated twice for each layout. Improvements in the second run may suggest that the layout is learnable and memorable which allows for better performance when users are more familiar with the layout. Each layout was also assigned a relative complexity metric based on a simplified version of Comber & Maltby (1995) screen evaluation methods, with 1 being minimum complexity and 4 being maximum complexity.

Grid Layout:	1 (Minimum Complexity)
50% + 23:77 Layout:	2 (Low Complexity)
Conformed Golden Spiral Layout:	3 (High Complexity)
Unaligned Layout:	4 (Maximum Complexity)

Users were then asked to rate these layouts in terms of attractiveness, ease of use and need for redesign or rearrangement from 1 to 5 as well as to sort them from best to worst.

8.1.3 Results

Based on the data obtained from this experiment *(which can be found at Appendix E of this report)*, the best layout can be observed in terms of time taken to finish all tasks, user ratings of the usability of the layout in performing given tasks and user preferences in layout selection. These information were tabulated in *Table 8.1.3* and presented in the following page.

	GL	FTL	CGSL	UL
Relative Layout Complexity	1	2	3	4
Total Time on 1st Run	64.09s	52.19s	69.54s	63.82s
Total Time on 2nd Run	56.05s	55.03s	67.69s	54.81s
Average Time on 1st Run	10.68s	8.67s	11.59s	10.64s
Average Time on 2nd Run	9.34s	9.17s	11.28s	9.14s
Total Average Time	10.01s	8.94s	11.44s	9.89s
Percentage of Errors on 1st Run	67%	0%	17%	83%
Percentage of Errors on 2nd Run	17%	33%	33%	33%
Average Percentage of Errors	42%	17%	25%	58%
Total Attractiveness Rating	26	27	17	13
Total Ease of Use Rating	27	28	15	13
Total Redesign Rating	-9	-9	-25	-24
Total Preference Rating	17	22	10	9
Total Satisfaction Rating	61	68	17	11

Table 8.1.3: Usability Experiment Results

8.1.3.1 Effectiveness

The effectiveness of a screen layout is determined by the percentage of errors made while performing the tasks outlined in this experiment. The higher the average percentage of errors, the lower the effectiveness of the layout. Errors are simplified into two values, 0 or 1. Any selection and/or typing errors made by the participants during this experiment will result in an error value of 1. These values are added together for both the first run and the second run to devise a percentage of errors per run. The percentage of errors are then averaged out to obtain an average percentage for each screen layout. These were plotted into a graph in *Figure 8.1.3.1* to examine its relationship.

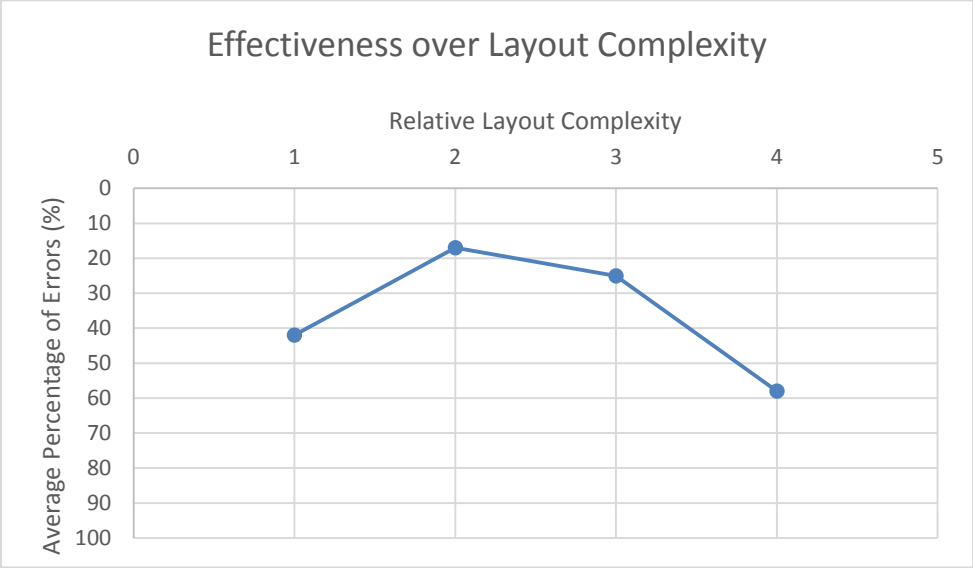


Figure 8.1.3.1: Effectiveness over Layout Complexity Graph

8.1.3.2 Efficiency

Efficiency is determined by the time taken in performing all 5 tasks outlined in this experiment. The lower the time taken, the higher the efficiency as shown in the *Figure 8.1.3.2* graph below.

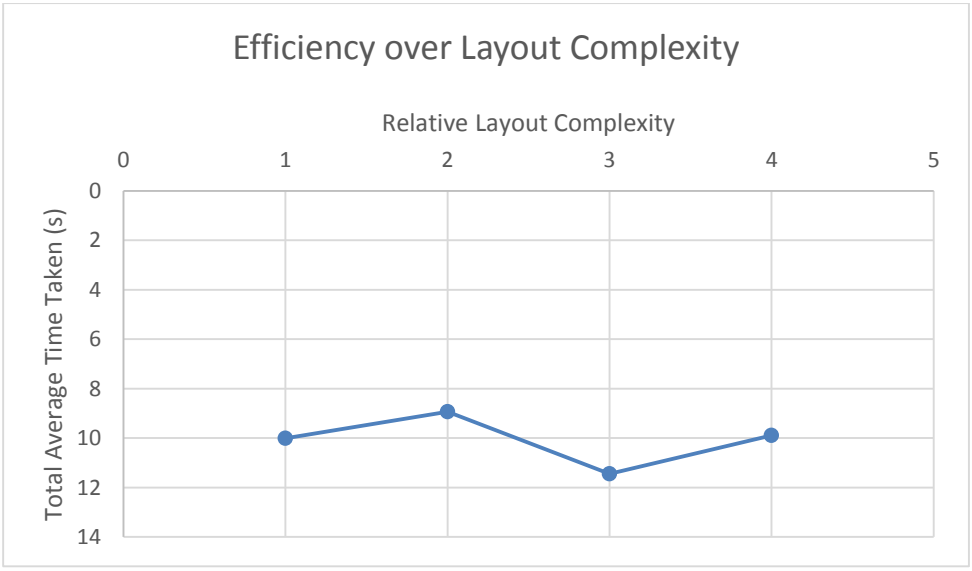


Figure 8.1.3.2: Efficiency over Layout Complexity Graph

8.1.3.3 Satisfaction

User satisfaction is divided into 4 parts in which users rate based on their opinion and subjective experiences. First criteria is attractiveness which is used to determine the users' point of view of the layout in terms of aesthetics alone. Attractiveness can be attributed to desirability or likeability. Second criteria is ease of use which is used to cognitively study user perception in task performance. Third criteria is the need for redesign or rearrangement which is used to determine acceptance of the layout because some users may dislike the appearance and usability of a screen layout but are still willing to accept and perform tasks as normal in the layout. User acceptance can be used to subjectively determine how much of an "eyesore" the layout is to look at; thus requiring the urge for rearrangement. An example would be where a user rates the CGSL layout badly in terms of aesthetics and usability, but because it maximizes the use of space, the user may feel less need to rearrange windows compared to the UL layout which has windows scattered on the screen with spaces in between. Lastly, the fourth criteria is preference from best to worst. Users are simply told to state their preference of the 4 layouts presented to them. All of these factors contribute towards overall satisfaction of the screen layout. *Figure 8.1.3.3* below plots this into a graph to examine the relationship between satisfaction and layout complexity.

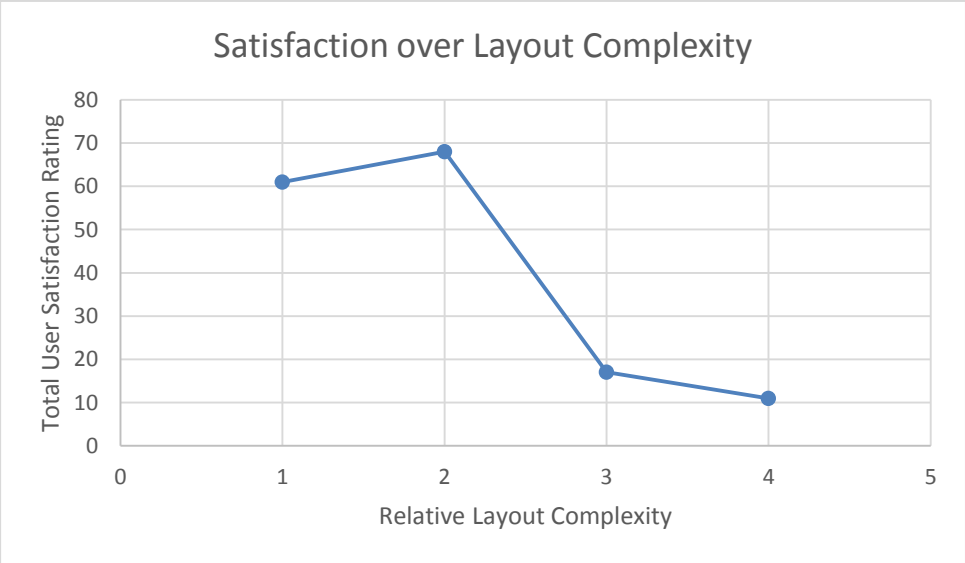


Figure 8.1.3.3: Satisfaction over Layout Complexity Graph

8.1.4 Discussion

Factors affecting this experiment consist of computer habits where some participants were observed to drag and drop to a presumed location first before actually moving to the actual location. This particular occurrence may be associated with a habit of positioning certain windows at certain parts of the screen in their home computer. Some participants were also observed to have a habit of dragging items from left-to-right which allowed them to perform much faster and naturally than dragging items from right-to-left.

Another factor observed which affects the time taken in performing tasks is the size of windows. Layouts such as the CGSL allowed for a very small window compared to other layouts. Participants were observed to have trouble navigating and finding items in smaller windows, even in the UL layout. For example, when dragging and selecting text in the web browser, participants scroll up and down to ensure that all of the text is selected because they are unable to see all of the text completely. Cognitively, a lot of focused attention is put into carrying out tasks within the small window and participants seem to be "lost" when returning to the screen to perform tasks with other windows.

Lastly, the familiarity of tasks may also influence the result for each layout. However, this factor is of the least concern because the FTL layout was presented to the participants first only after a single trial run, yet it was the best performer in all usability aspects compared to the other layouts.

8.1.5 Conclusion

Throughout the different aspects of usability different relationships were observed based on relative layout complexity. Notably, the effectiveness of a screen layout over layout complexity generated a parabolic curve similar to the predicted usability over complexity graph in Comber & Maltby's (1995) experiment. Relationships for other factors such as efficiency are difficult to determine due to the lack of data but was observed to have the appearance of a sinusoidal graph. Lastly, user satisfaction is the least predictable and no clear data relationships can be determined.

Overall, the FTL layout with a low layout complexity performs the best in attractiveness, ease of use and satisfaction. The two extremes of complexity – GL being the minimum complexity and UL being the maximum complexity, performed worse than the FTL low complexity layout. In general, low complexity layouts are significantly more favourable than high complexity layouts.

8.2 White-Box Testing

The white-box testing is done at the final iteration of the development process to find and fix programming errors and bugs that occur during run-time. The test explores different paths along the user interactions defined in the Use Case Diagram with various types of input and checks if the output is correct. It is done using a test plan outlined in *Appendix F* of this report. The test has uncovered several bugs during run-time, mostly due to logic errors.

8.2.1 Bugs Found & Fixed

1. **Hidden windows did not show properly.**
   Solution: Fix hide logic.
   Hidden windows were removed from the category upon update, so the application had nothing to show. Hidden windows are now added to a hidden group for reference.

2. **Only half of the non-existent windows were removed when updating a category.**
   Solution: Omit loop counter increment.
   When refreshing the categories to remove non-existent windows, the loop ends halfway through, only removing half of the windows each time. This was fixed, due to the dynamic nature of Lists which move the objects up to fill in empty spaces. When an object is removed, the next object shifts to the current index. When the loop counter increases and checks for the next index, the object is no longer there.

8.2.2 Unfixed Bugs

By default, the taskbar in Windows 7 is docked at the bottom of the screen. However, when docking and arranging windows, windows are seen to be positioned behind the taskbar if the taskbar is docked to the left. This problem does not occur when the taskbar is at the bottom, right or top of the screen. This is because the SetWindowPos function sets window positions from the leftmost position of the screen (x = 0). In order to fix this, an offset must be added specifically for this situation. Due to time constraints, this bug was not fixed because it requires more work than predicted. The solution requires the application to not only obtain the taskbar position, but also its size because taskbars can be freely resized in Windows 7. This also requires an implementation of another API, which is the Shell32 API to get taskbar information.

8.3 Heuristics and Keystroke-Level Model

Heuristics Check	Yes	No	N/A	Comments
Visibility of System Status The system should always keep user informed about what is going on, through suitable feedback within reasonable time.	✓			Interface has labels to keep track of status, user is shown a balloon tooltip when changing states.
Match Between System and Real World Arrange information in a natural and logical order with language, words and concepts familiar to the user.	✓			No technical jargon used, windows and category buttons are listed from top-to-bottom and left-to-right.
User Control and Freedom Allow users to leave the unwanted state without going through a long dialogue.	✓			Can easily change states and tells the user how to do so with tooltips.
Consistency and Standards Follow platform conventions, consistent meanings without conflicting usage patterns.	✓			Consistent interface, icons make sense and have tooltips for accessibility.
Error Prevention Eliminate error-prone conditions or present users with a confirmation before committing to an action with irreversible consequences.	✓			Always asks for confirmation on irreversible actions.
Recognition Rather Than Recall Minimize user memory load with visible objects, actions and options as well as easily retrievable instructions or descriptions.	✓			Icons and visual representations are easily understandable and have tooltips as reminders.
Flexibility and Efficiency of Use Allow users to tailor frequent actions and have accelerators not visible to beginners to speed interactions for expert users.	✓			Most functions accessible with both mouse and keyboard, interface helps beginner users a lot.
Aesthetic and Minimalist Design Does not contain information and visual loads which are irrelevant.	✓			Simple grid interface with mostly icons. Tooltips only show when needed.
Help Users Recognize, Diagnose and Recover from Errors Error messages explained in plain language, precisely indicate the problem and constructively suggest a solution.	✓			Tells the user why the problem occurred and what should be done. Also tells users how to exit from unwanted states.
Help and Documentation Provide easy-to-find information focused on user tasks, list precise instructions and documentation must not be too large.	✓			Clear and easy instructions with tooltips on each button and label.

Table 8.3a: Application's Heuristics Checklist

Description	Operation	Time (sec)
Mentally prepare for auto-dock hotkey	M[hotkey]	1.20
Hand to keyboard	H[keyboard]	0.40
Hotkey Ctrl + Shift + Tab	K[hotkey]	0.20
Mentally prepare for docking keys	M[hotkey]	1.20
Hand to arrow keys	H[keyboard]	0.40
Hold Alt Key	K[key]	0.20
Tap left arrow key	K[key]	0.20
Tap right arrow key	K[key]	0.20
Tap right arrow key	K[key]	0.20
	TOTAL ESTIMATED TIME:	4.20
KLM Formula: 2(M + H) + 5K = 4.2 seconds		

Table 8.3b: Application's Keystroke Level Model

The real problem with most window management tool is the lack of automation. Users have to manually select windows one-by-one and iterate the same actions over several windows. The application developed in this project identifies this problem and proposes window groups and automatic switching as a solution. As seen in the KLM analysis in *Table 8.3b,* all point-and-click operations were eliminated by automatically switching to the window located at the back-most position of the screen. This greatly reduces the number and type of operations, significantly increasing task efficiency. In addition, due to user-centered design methods for flexibility, this automatic switching can simply be ignored if the user does not want to switch windows by letting go the Alt key.

The heuristics check in *Table 8.3a* ensures high usability in all of its functionality. For example, the application makes use of balloon tooltips instead of message dialogs to show acknowledgement messages to allow the user to know the current system state; thus, it never interrupts the user with the exception of error messages and confirmation messages which uses dialogs. Other examples include options to run the application on startup, renaming of categories, automatic arrangement and adding of windows into categories, and other features which offer total control over the interface.

8.4 Limitations of the Current System

The project scope was focused on solving Windows 7 window management problems. The first and foremost limitation is that the application is currently only suited to work in Windows 7. Although Windows 8 may use similar technologies, the application was not tested in Windows 8 and new bugs may occur under that environment.

Due to time constraints, a functionality which was scheduled to complete within the project timeframe were not fully developed. The application aimed to implement reconfigurable hotkeys for user-friendliness but it was dropped from the project due to its low MoSCoW priority within the tight timeframe. Configuring hotkeys require persistent data to be stored in the user's computer file system and was well achieved using XML technology in the project. However, there was not enough time to create an actual interface for users to change and customize hotkeys through the application. Implementation of an interface for binding hotkeys requires more time than intended for data manipulation and file input/output management. It is possible however, to change the hotkeys directly in the XML file because the application still reads hotkey bindings from the XML file.

As seen in the White-Box Testing, a bug on window positioning was also found for users with taskbars docked on the left side of the screen. For now, the application does not work correctly in this particular situation. Future implementations will look more into taskbar information and other screen elements which may affect the application's arrangement functionality.

Other future implementations include:

- Animated window movements
- Compatibility for other versions of Windows
- Compatibility for other platforms

Overall, the application has met project requirements and satisfied its primary functions as an efficient solution for window management limitations.

9.0 Conclusion

Window manipulation coupled with desktop space management using hotkeys and usable screen layouts increases multi-tasking effectiveness. The grouping of windows into categories serves as an effective window manipulation method by mapping real-life methods into computer-based operations. Manipulating windows by groups reduces the operation of selecting desired windows one-by-one, significantly reducing preparation time, number of task iterations per window, as well as completely removing the lengthy point-and-click operation in finding desired windows. In addition, the arrangement of windows on screen affects human-computer interaction and usability due to the cognitive tendency of humans to focus on one window at a time when multi-tasking. Window arrangements such as the 50% + 23:77 layout which allow users to focus on a large main window while providing functionality to allow switching window positions with the main window increases multi-tasking capabilities dramatically. The cognitive psychology of humans was analyzed using KLM-GOMS analysis to further reveal that hotkey operations can be simplified by allowing them to be executed with one hand, reducing estimated time taken by 0.40 seconds for each hand movement operation from keyboard to mouse and vice versa. The simplification of hotkey operations is crucial for frequent tasks in order to save time.

As a result of applied theory, standards, analysis and experimentation studied in this project, a highly usable working application was developed using Visual C#, .NET Framework 4.0, Win32 API and DWM API to address the limitations of window manipulation in Windows 7 as well as increasing multi-tasking capabilities and efficiencies through space management of windows on computer screens. Future usability improvements for the software include animation of window movement, customizable layouts and compatibility with other platforms and operating systems.

In terms of user preference, it is found that users mostly prefer layouts with low complexity, followed by minimal complexity, high complexity and maximum complexity. However, the feedback on user preferences were obtained from only 6 participants; therefore, its results may not be completely reliable and a larger sample size should be used in future.

10.0 Bibliography

Actual Tools. (2013). *Actual Window Manager: Multiple Monitors, Virtual Desktops, Windows Control and Other Useful Tools*. [online] Available at: http://www.actualtools.com/windowmanager/ [Accessed 7 Nov 2013].

Benton, N., Kennedy, A. & Russo, C. (2004). Adventures in Interoperability: The SML.NET Experience. In: *PPDP '04*. pp.10-12.

Brooks, E. (2009). *Token by brsev on deviantART*. Available at: http://brsev.deviantart.com/art/Token-128429570 [Accessed: 21 Jan 2014].

Bonsiepe, G. A. (1968). *A method of quantifying order in typographic design*. Journal of Typographic Research, 2, 203-220.

Boselie, F. (1997). *The golden section and the shape of objects*. Empirical Studies of the Arts, 15, 131-141.

Card, S. K., Moran, T. P. & Newell, A. (1983). *The Psychology of Human-computer Interaction*. Hillsdale, N.J.: L. Erlbaum Associates.

Chang, D., Dooley, L. & Tuovinen, J.E. (2002). *Gestalt Theory in Visual Screen Design - A New Look at an Old Subject*. In Proc. WCCE2001 Australian Topics: Selected Papers from the Seventh World Conference on Computers in Education, Copenhagen, Denmark. CRPIT, 8. McDougall, A., Murnane, J. and Chambers, D., Eds. ACS. 5-12.

Chen, R. (2010). *Which windows appear in the Alt+Tab list? - The Old New Thing, MSDN Blogs*. Available at: http://blogs.msdn.com/b/oldnewthing/archive/2007/10/08/5351207.aspx [Accessed: 18 Dec 2013].

Comber, T. & Maltby, J. R. (1995). *Evaluating usability of screen designs with layout complexity*. [online] Southern Cross University. Available at: http://epubs.scu.edu.au/cgi/viewcontent.cgi?article=1000&context=comm_pubs [Accessed: 10 Nov 2013].

Davis, S. T. & Jahnke J. C. (1991). *Unity and the golden section: Rules for aesthetic choice?*. American Journal of Psychology, 104, 257-277.

De Smet, B. (2006). *Programming the Windows Vista DWM in C#*. [online] B# .NET Blog. Available at: http://bartdesmet.net/blogs/bart/archive/2006/10/05/4495.aspx [Accessed 22 Feb 2014].

Dinham, P. (2011). *c++ - Why does EnumWindows return more windows than I expected?, Stack Overflow*. Available at: http://stackoverflow.com/questions/7277366/why-does-enumwindows-return-more-windows-than-i-expected [Accessed: 18 Dec 2013].

Hartikainen, J. (2007). *Find an application's icon with WinAPI*. [online] CodeUtopia. Available at: http://codeutopia.net/blog/2007/12/18/find-an-applications-icon-with-winapi/ [Accessed 22 Feb 2014].

Hatton, S. (2008). *Choosing the right prioritisation method*. 19th Australian Conference on Software Engineering, 517 - 526.

Johnson, A. (1999). *Famous problems and their mathematicians*. Englewood, Colo.: Teacher Ideas Press.

Kerr, J. & Hunter, R. (1994). *Inside RAD: How to Build a Fully Functional System in 90 Days or Less*. 1st ed. New York: McGraw-Hill.

Kieras, D. E. (1993). *Using the Keystroke-Level Model to Estimate Execution Times*. The University of Michigan. Available at: http://www.pitt.edu/~cmlewis/KSM.pdf [Accessed 8 Nov 2013].

Lancet, Y. (2013). *Review: Chameleon Window Manager has many features and a few too many bugs | PCWorld*. [online] Available at: http://www.pcworld.com/article/2036721/review-chamelon-window-manager-has-many-features-and-a-few-too-many-bugs.html [Accessed: 9 Nov 2013].

Lane, D. M., Napier, H. A., Peres, S. C. & Sandor, A. (2004). *Hidden costs of graphical user interfaces: Failure to make the transition from menus and icon toolbars to keyboard shortcuts*. International Journal of Human–Computer Interaction, 18, 133–144.

Lipsum Generator. (2014). *Lorem Ipsum - All the facts*. Available at: http://www.lipsum.com/ [Accessed: 12 Mar 2014].

Ma, Q. (2009). *The Effectiveness of Requirements Prioritization Techniques for a Medium to Large Number of Requirements: A Systematic Literature Review*. [online] AUT Scholarly Commons. Available at: http://aut.researchgateway.ac.nz/bitstream/handle/10292/833/MaQ.pdf?sequence=3 [Accessed 4 Oct 2013].

Microsoft Windows. (2013). *Keyboard shortcuts - Microsoft Windows Help*. [online] Available at: http://windows.microsoft.com/en-my/windows/keyboard-shortcuts#keyboard-shortcuts=windows-7 [Accessed 7 Nov 2013].

Microsoft Windows (2013). *Manage multiple windows - Microsoft Windows Help*. [online] Available at: http://windows.microsoft.com/en-my/windows/manage-multiple-windows#1TC=windows-7&section_4 [Accessed 7 Nov 2013].

NeoSoft Tools. (2013). *Chameleon Window Manager*. [online] Available at: http://www.chameleon-managers.com/window-manager/ [Accessed 7 Nov 2013].

Ngo D. & Ch'ng E. (2001). *Screen Design: composing with dynamic symmetry*. Displays, No. 22, pp. 115-124.

NTWind Software. (2013). *WindowSpace - Organize Your Desktop Workspace for More Comfort with WindowSpace*. [online] Available at: http://www.ntwind.com/software/windowspace.html [Accessed 7 Nov 2013].

Rutland, C. (2010). *Global Hotkeys - C# Tutorials*. [online] Dream.In.Code. Available at: http://www.dreamincode.net/forums/topic/180436-global-hotkeys/ [Accessed 3 Mar 2014].

Tognazzini, B. (1987). *Keyboard vs. The Mouse*. [online] Available at: http://www.asktog.com/TOI/toi06KeyboardVMouse1.html [Accessed: 7 Nov 2013].

Tudor, D. & Walter, G. A. (2006). Using an agile approach in a large, traditional organisation. *Proceedings of AGILE 2006 Conference (AGILE'06),* 367-373.

Tullis, T. S. (1983). *The formatting of alphanumeric displays: a review and analysis*. Human Factors, 25(6), 557-582.

Peres, S. C., Tamborello II, F. P., Fleetwood, M. D., Chung, P. & Paige-Smith, D. L. (2004). *Keyboard Shortcut Usage: The Roles of Social Factors and Computer Experience*. [online] Houston, TX: Department of Psychology, Rice University. Available at: http://chil.rice.edu/research/pdf/PeresEtal-HFES.pdf [Accessed: 9 Nov 2013].

van Schaik, P. & Ling, J. (2003). *The effects of screen ratio and order on information retrieval in web pages.* Displays, 24 (4-5), pp.187-195.

Appendix A – Chameleon Window Manager Heuristics & KLM

Heuristics Check	Yes	No	N/A	Comments
Visibility of System Status The system should always keep user informed about what is going on, through suitable feedback within reasonable time.	✓			Visual cues are good enough to represent current system state.
Match Between System and Real World Arrange information in a natural and logical order with language, words and concepts familiar to the user.		✓		Technical terms were used, incomprehensible to an average user.
User Control and Freedom Allow users to leave the unwanted state without going through a long dialogue.	✓			Can easily change states.
Consistency and Standards Follow platform conventions, consistent meanings without conflicting usage patterns.		✓		Interface and layout is not consistent, all over the place.
Error Prevention Eliminate error-prone conditions or present users with a confirmation before committing to an action with irreversible consequences.		✓		Does not eliminate error-prone conditions.
Recognition Rather Than Recall Minimize user memory load with visible objects, actions and options as well as easily retrievable instructions or descriptions.	✓			Icons and visual representations are easily understandable.
Flexibility and Efficiency of Use Allow users to tailor frequent actions and have accelerators not visible to beginners to speed interactions for expert users.		✓		No use of hotkeys, all users must mouse over to button to perform functions.
Aesthetic and Minimalist Design Does not contain information and visual loads which are irrelevant.		✓		Too many buttons added to all windows.
Help Users Recognize, Diagnose and Recover from Errors Error messages explained in plain language, precisely indicate the problem and constructively suggest a solution.		✓		Some functions did not work as expected and no suggestions were made to fix the problem.
Help and Documentation Provide easy-to-find information focused on user tasks, list precise instructions and documentation must not be too large.	✓			Clear and easy instructions.

Table A1: Heuristics Checklist for Chameleon Window Manager

Description	Operation	Time (sec)
Reach for mouse	H[mouse]	0.40
Point to desired window	P[window]	1.10
Select window	B[mouse]	0.10
Move cursor to layout button	P[button]	1.10
Click layout button	B[mouse]	0.10
Move cursor to desired layout	P[layout]	1.10
Mentally prepare by looking at layout set	M[layout]	1.20
Click layout	B[mouse]	0.10
Point to desired window	P[window]	1.10
Select window	B[mouse]	0.10
Move cursor to layout button	P[button]	1.10
Click layout button	B[mouse]	0.10
Move cursor to desired layout	P[layout]	1.10
Click layout	B[mouse]	0.10
Point to desired window	P[window]	1.10
Select window	B[mouse]	0.10
Move cursor to layout button	P[button]	1.10
Click layout button	B[mouse]	0.10
Move cursor to desired layout	P[layout]	1.10
Click layout	B[mouse]	0.10
	TOTAL ESTIMATED TIME:	12.40
KLM Formula: M + H + 9(P + B) = 12.4 seconds		

Table A2: Keystroke Level Model for Chameleon Window Manager

Appendix B – WindowSpace Heuristics & KLM

Heuristics Check	Yes	No	N/A	Comments
Visibility of System Status				
The system should always keep user informed about what is going on, through suitable feedback within reasonable time.	✓			Visual cues are good enough to show current status.
Match Between System and Real World				
Arrange information in a natural and logical order with language, words and concepts familiar to the user.		✓		Technical terms were used but not explained.
User Control and Freedom				
Allow users to leave the unwanted state without going through a long dialogue.	✓			Can easily change states.
Consistency and Standards				
Follow platform conventions, consistent meanings without conflicting usage patterns.	✓			Consistent layout with standardized spacing and well-structured presentation.
Error Prevention				
Eliminate error-prone conditions or present users with a confirmation before committing to an action with irreversible consequences.		✓		Does not show confirmation messages for irreversible actions.
Recognition Rather Than Recall				
Minimize user memory load with visible objects, actions and options as well as easily retrievable instructions or descriptions.	✓			Clear and concise instructions for all functions.
Flexibility and Efficiency of Use				
Allow users to tailor frequent actions and have accelerators not visible to beginners to speed interactions for expert users.	✓			Easy to use with reconfigurable hotkeys.
Aesthetic and Minimalist Design				
Does not contain information and visual loads which are irrelevant.	✓			Clean interface with only relevant information, nothing else.
Help Users Recognize, Diagnose and Recover from Errors				
Error messages explained in plain language, precisely indicate the problem and constructively suggest a solution.	✓			Constructive error messages.
Help and Documentation				
Provide easy-to-find information focused on user tasks, list precise instructions and documentation must not be too large.	✓			Well-structured and well-designed support documentation, high readability.

Table B1: Heuristics Checklist for WindowSpace

Description	Operation	Time (sec)
Reach for mouse	H[mouse]	0.40
Move cursor to desired window	P[window]	1.10
Select window	B[mouse]	0.10
Mentally prepare for arrangement hotkey set	M[hotkey]	1.20
Hand to keyboard	H[keyboard]	0.40
Hotkey Win + Num 4	K[hotkey]	0.20
Reach for mouse	H[mouse]	0.40
Point to next desired window	P[window]	1.10
Select window	B[mouse]	0.10
Hand to keyboard	H[keyboard]	0.40
Hotkey Win + Num 9	K[hotkey]	0.20
Reach for mouse	H[mouse]	0.40
Switch to next window	P[window]	1.10
Select window	B[mouse]	0.10
Hand to keyboard	H[keyboard]	0.40
Hotkey Win + Num 3	K[hotkey]	0.20
	TOTAL ESTIMATED TIME:	7.80
KLM Formula: M + 3(2H + P + B + K)= 7.8 seconds		

Table B2: Keystroke Level Model for WindowSpace

Appendix C – Actual Window Manager Heuristics & KLM

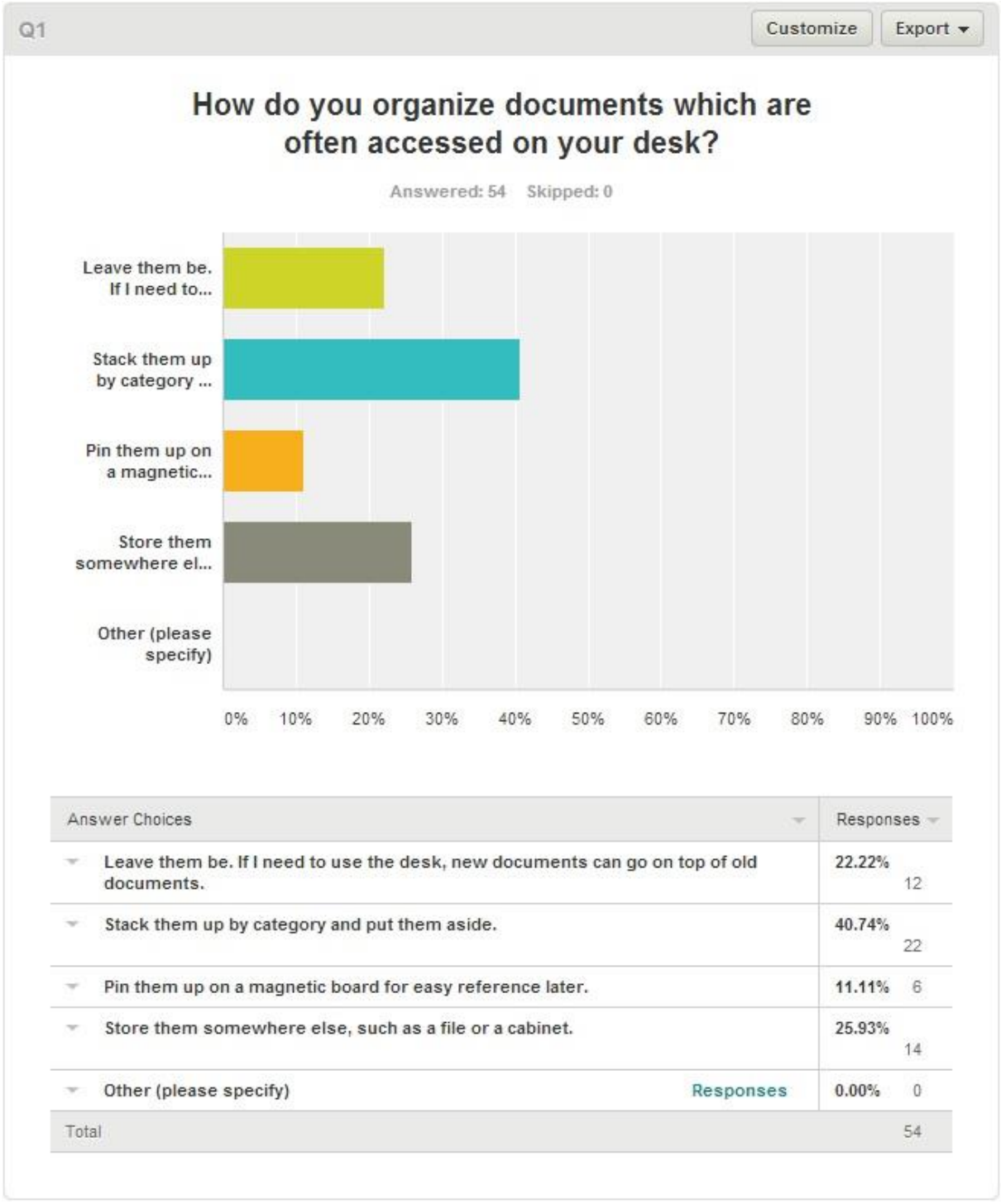
Heuristics Check	Yes	No	N/A	Comments
Visibility of System Status				Visual cues are good enough to show current status.
The system should always keep user informed about what is going on, through suitable feedback within reasonable time.	✓			
Match Between System and Real World				Suitable language with no technical jargon.
Arrange information in a natural and logical order with language, words and concepts familiar to the user.	✓			
User Control and Freedom				Can easily change states.
Allow users to leave the unwanted state without going through a long dialogue.	✓			
Consistency and Standards				Consistent flow and very well-structured layout.
Follow platform conventions, consistent meanings without conflicting usage patterns.	✓			
Error Prevention				Asks user for confirmation before performing irreversible actions.
Eliminate error-prone conditions or present users with a confirmation before committing to an action with irreversible consequences.	✓			
Recognition Rather Than Recall				Too lengthy instructions and hard to retrieve them, information overload.
Minimize user memory load with visible objects, actions and options as well as easily retrievable instructions or descriptions.		✓		
Flexibility and Efficiency of Use				Does not provide faster interaction for many functions.
Allow users to tailor frequent actions and have accelerators not visible to beginners to speed interactions for expert users.		✓		
Aesthetic and Minimalist Design				Lots of irrelevant information which can be further shortened.
Does not contain information and visual loads which are irrelevant.		✓		
Help Users Recognize, Diagnose and Recover from Errors				Detailed error messages.
Error messages explained in plain language, precisely indicate the problem and constructively suggest a solution.	✓			
Help and Documentation				Extremely lengthy documentation, very steep learning curve.
Provide easy-to-find information focused on user tasks, list precise instructions and documentation must not be too large.		✓		

Table C1: Heuristics Checklist for Actual Window Manager

Description	Operation	Time (sec)
Reach for mouse	H[mouse]	0.40
Move cursor to desired window	P[window]	1.10
Select window	B[mouse]	0.10
Hand to keyboard	H[keyboard]	0.40
Mental preparation for alignment hotkey set	M[hotkey]	1.20
Hotkey Win + Num 7	K[hotkey]	0.20
Mental preparation for resizing hotkey set	M[hotkey]	1.20
Hotkey Win + Ctrl + Num 0	K[hotkey]	0.20
Reach for mouse	H[mouse]	0.40
Move cursor to desired window	P[window]	1.10
Select window	B[mouse]	0.10
Hand to keyboard	H[keyboard]	0.40
Hotkey Win + Num 9	K[hotkey]	0.20
Mental preparation for resizing hotkey set #2	M[hotkey]	1.20
Hotkey Win + Ctrl + Num 5	K[hotkey]	0.20
Reach for mouse	H[mouse]	0.40
Move cursor to desired window	P[window]	1.10
Select window	B[mouse]	0.10
Hand to keyboard	H[keyboard]	0.40
Hotkey Win + Num 3	K[hotkey]	0.20
Hotkey Win + Ctrl + Num 5	K[hotkey]	0.20
	TOTAL ESTIMATED TIME:	10.80
KLM Formula: 3(M + 2H + 2K + P + B) = 10.8 seconds		

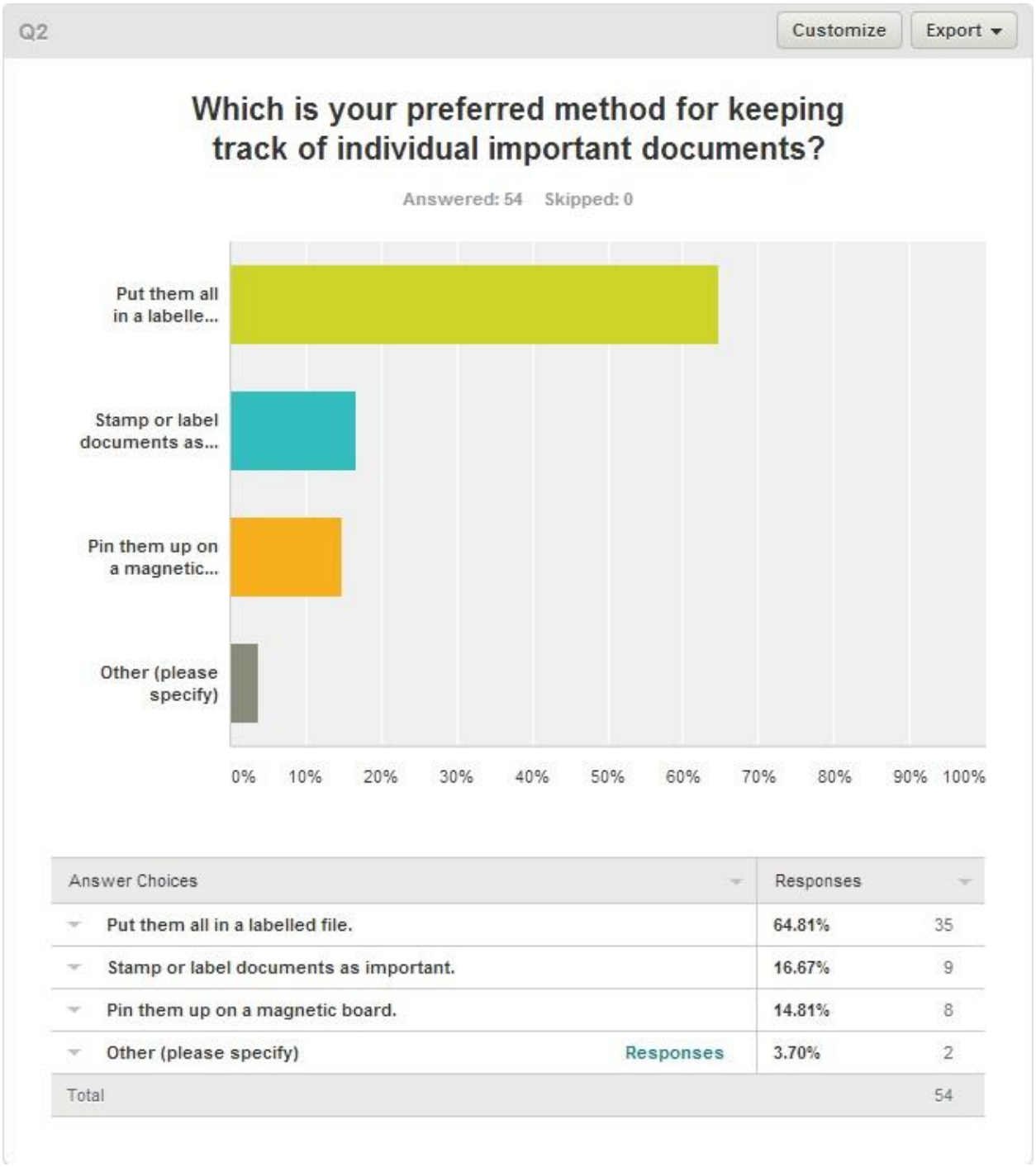
Table C2: Keystroke Level Model for Actual Window Manager

Appendix D – Questionnaire Results: Managing Documents and Windows



Answer Choices	Responses	
Leave them be. If I need to use the desk, new documents can go on top of old documents.	22.22%	12
Stack them up by category and put them aside.	40.74%	22
Pin them up on a magnetic board for easy reference later.	11.11%	6
Store them somewhere else, such as a file or a cabinet.	25.93%	14
Other (please specify) Responses	0.00%	0
Total		54

Figure D1: Question 1



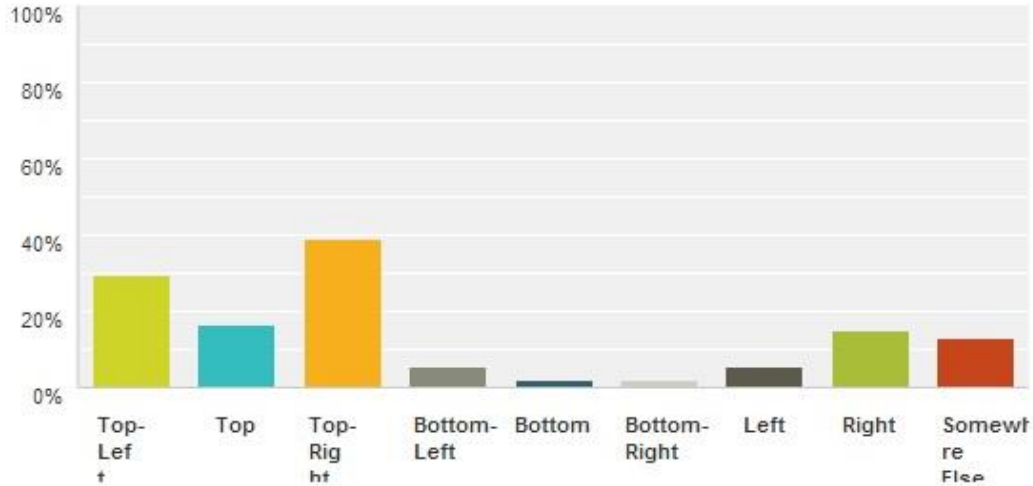
Answer Choices		Responses	
Put them all in a labelled file.		64.81%	35
Stamp or label documents as important.		16.67%	9
Pin them up on a magnetic board.		14.81%	8
Other (please specify)	Responses	3.70%	2
Total			54

Figure D2: Question 2

Q3 Customize Export

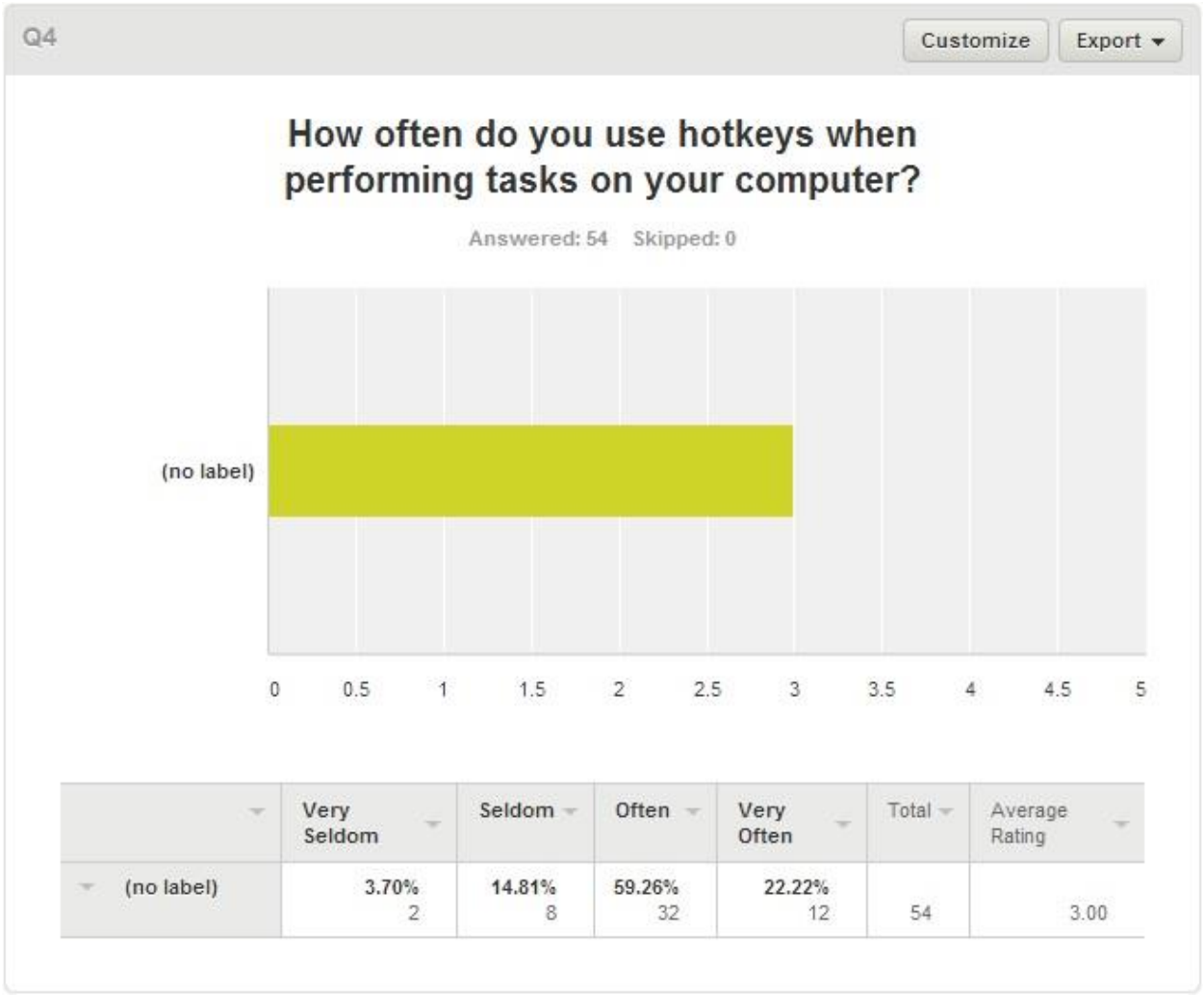
When working with documents on your desk, where do you usually put your stationery?

Answered: 54 Skipped: 0



Answer Choices	Responses	
Top-Left	29.63%	16
Top	16.67%	9
Top-Right	38.89%	21
Bottom-Left	5.56%	3
Bottom	1.85%	1
Bottom-Right	1.85%	1
Left	5.56%	3
Right	14.81%	8
Somewhere Else	12.96%	7
Total Respondents: 54		

Figure D3: Question 3



	Very Seldom	Seldom	Often	Very Often	Total	Average Rating
(no label)	3.70% 2	14.81% 8	59.26% 32	22.22% 12	54	3.00

Figure D4: Question 4

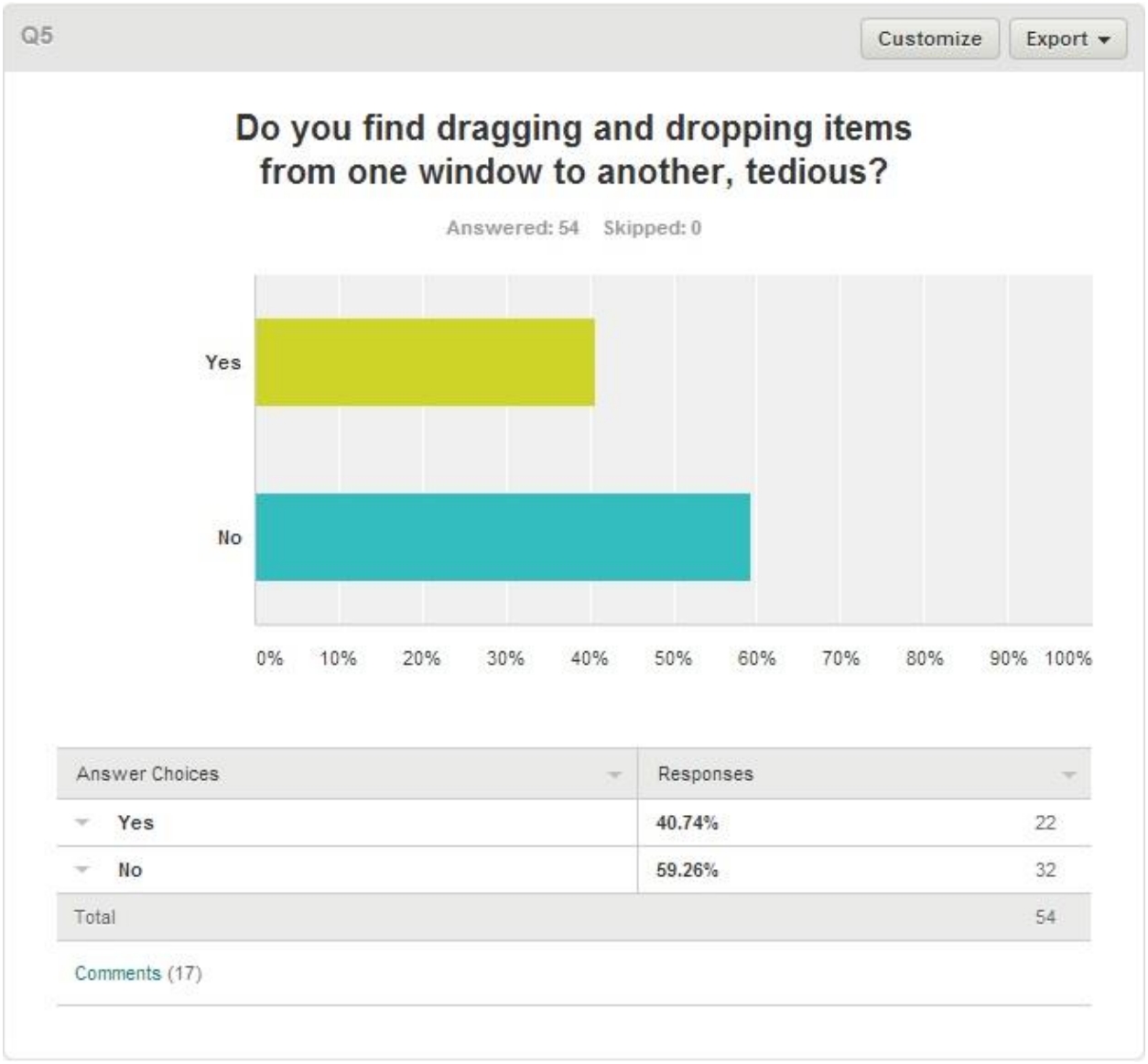


Figure D5: Question 5

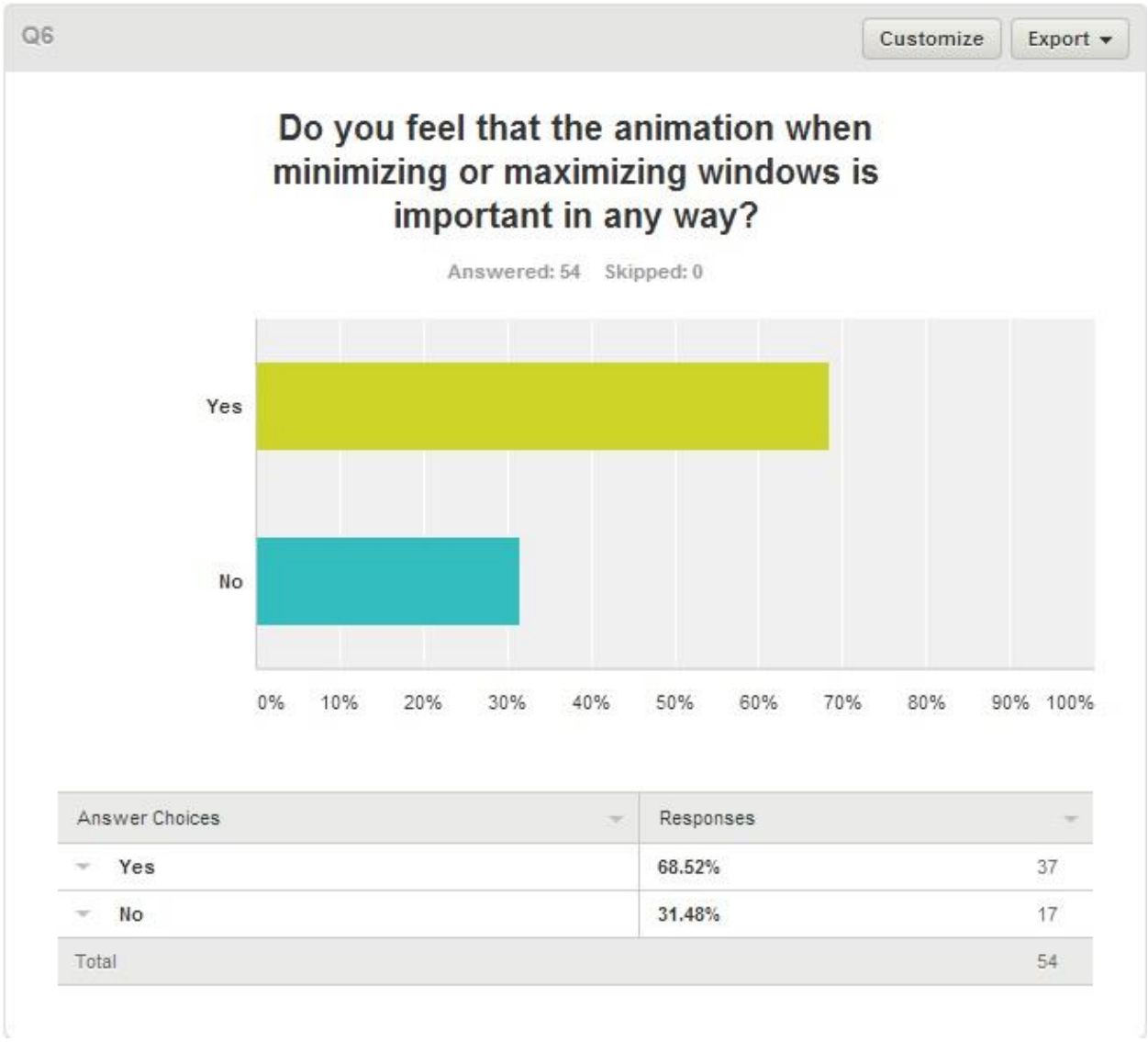
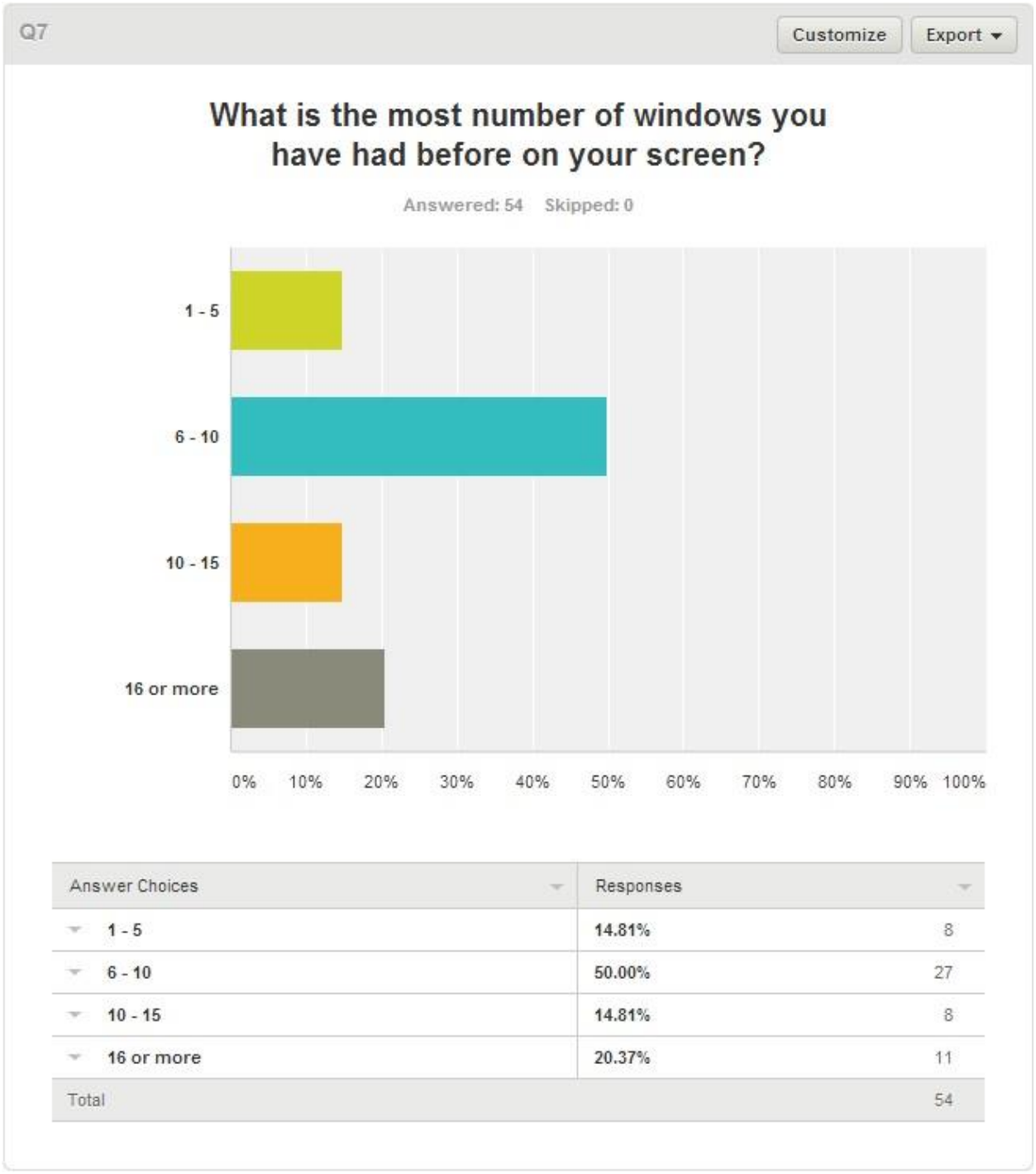
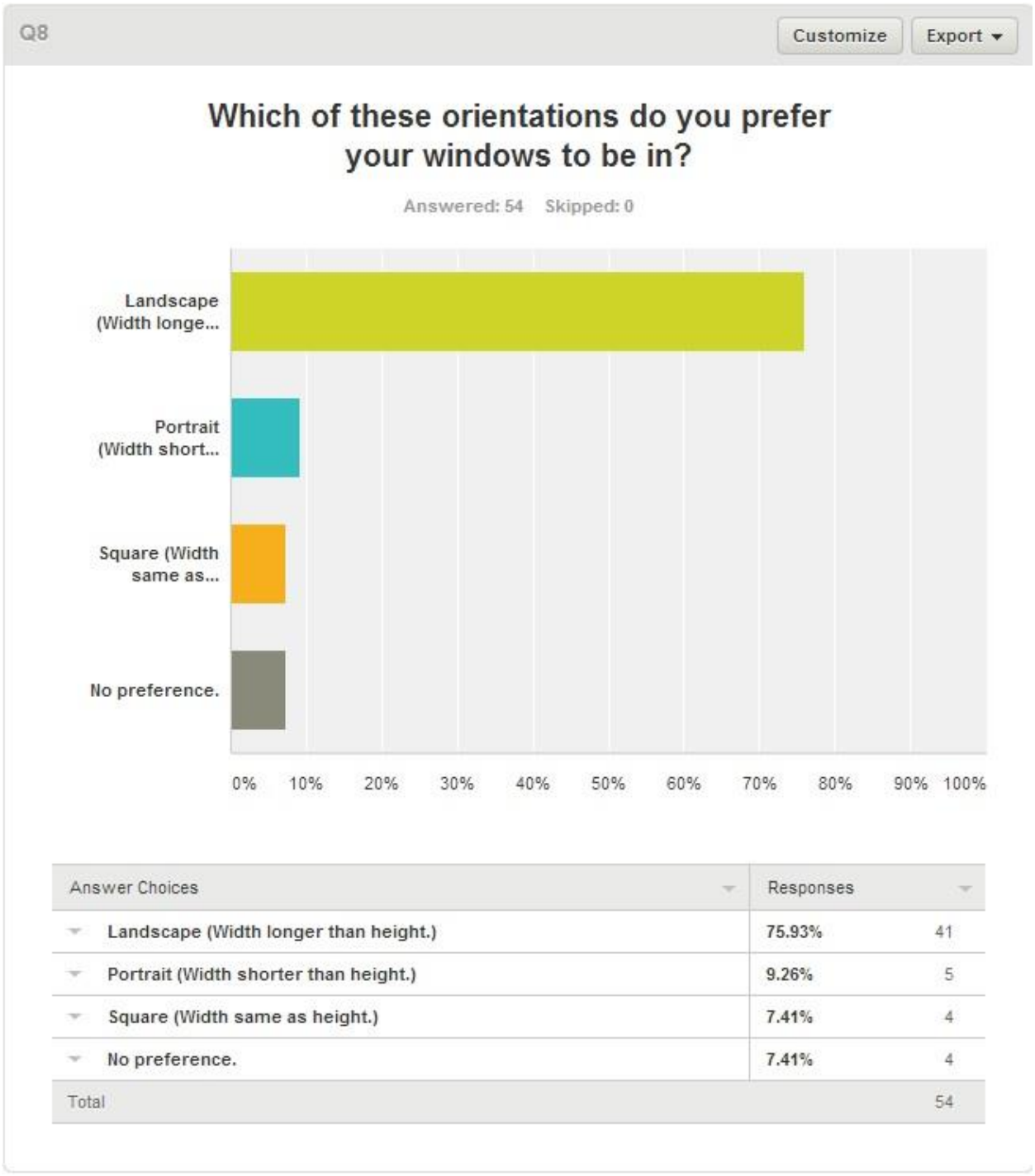


Figure D6: Question 6



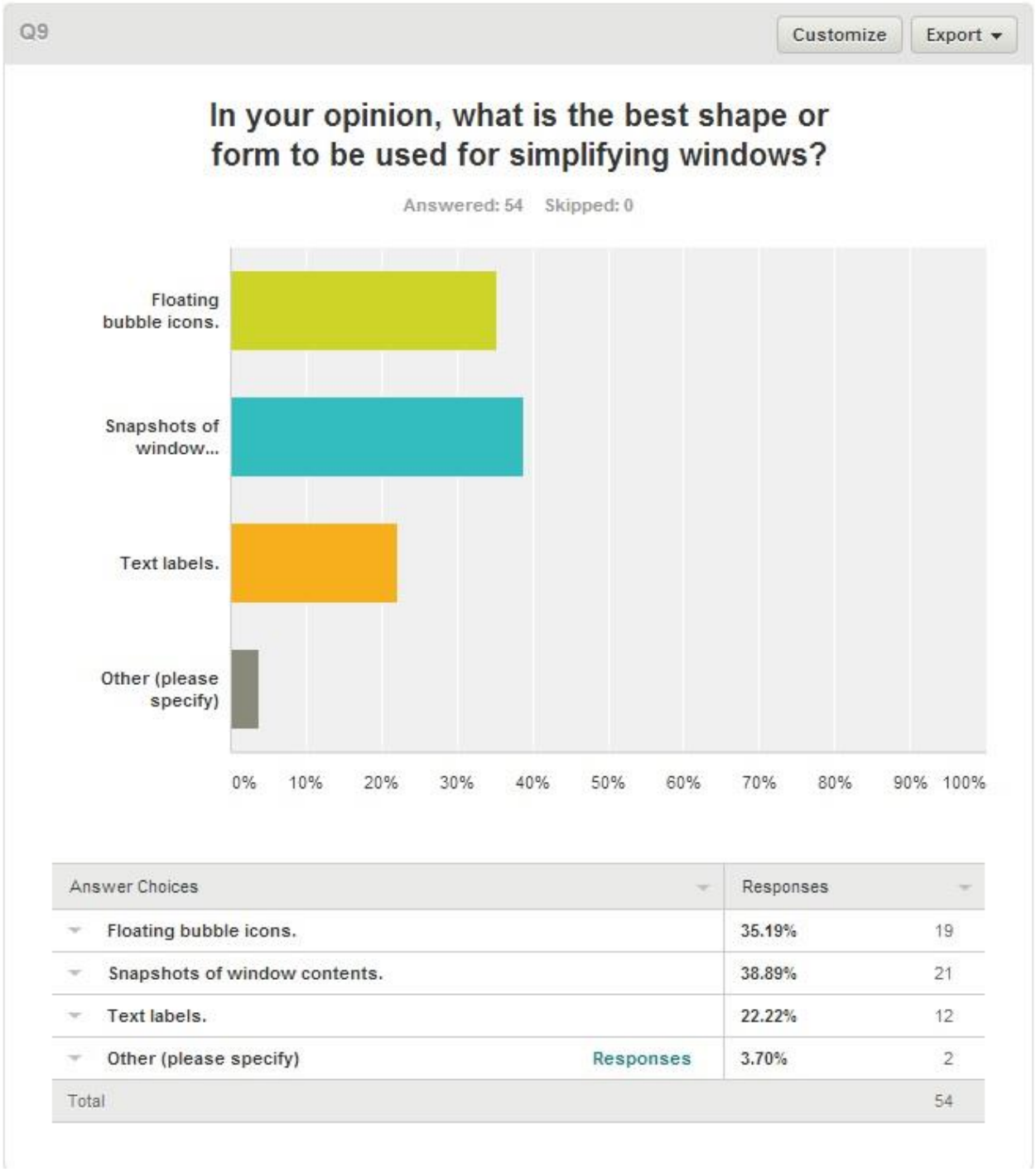
Answer Choices	Responses	
1 - 5	14.81%	8
6 - 10	50.00%	27
10 - 15	14.81%	8
16 or more	20.37%	11
Total		54

Figure D7: Question 7



Answer Choices	Responses	
Landscape (Width longer than height.)	75.93%	41
Portrait (Width shorter than height.)	9.26%	5
Square (Width same as height.)	7.41%	4
No preference.	7.41%	4
Total		54

Figure D8: Question 8



Answer Choices		Responses	
Floating bubble icons.		35.19%	19
Snapshots of window contents.		38.89%	21
Text labels.		22.22%	12
Other (please specify)	Responses	3.70%	2
Total			54

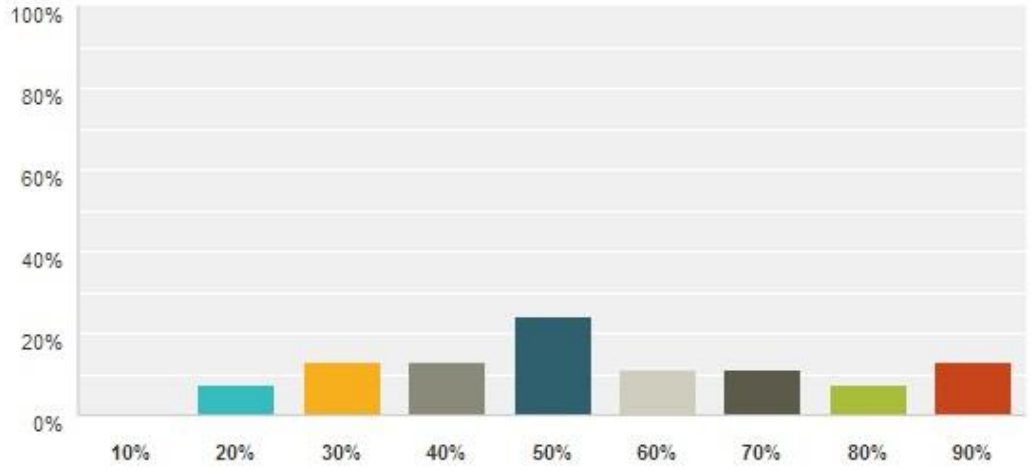
Figure D9: Question 9

Q10

Customize Export

For a window which is not fully maximized, what minimum percentage of the screen should it cover so that you feel comfortable working with it?

Answered: 54 Skipped: 0



Answer Choices	Responses	
10%	0.00%	0
20%	7.41%	4
30%	12.96%	7
40%	12.96%	7
50%	24.07%	13
60%	11.11%	6
70%	11.11%	6
80%	7.41%	4
90%	12.96%	7
Total		54

Figure D10: Question 10

Appendix E – Usability Experiment Data

Usability Experiment						
Course Name: XXXXXXXX – XXXXXXXXXXX						
System Name: Window Management Tool			**Version:** v0.1 Prototype (1st Iteration)			
	User #1	**User #2**	**User #3**	**User #4**	**User #5**	**User #6**
User Name	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX	XXXXX
50% + 23:77 Layout (FTL)						
	First Run					
Time Taken	7.2s	10.3s	9.0s	9.59s	5.6s	10.5s
Made an Error?	0	0	0	0	0	0
	Second Run					
Time Taken	6.4s	12.9s	7.8s	11.23s	5.9s	10.8s
Made an Error?	1	0	0	1	0	0
	Feedback (Rating Out of 5)					
Attractiveness	4	4	5	4	5	5
Ease of Use	5	4	5	4	5	5
Needs Redesign	1	2	2	2	1	1
Conformed Golden Spiral Layout (CGSL)						
	First Run					
Time Taken	10.2s	16.8s	11.0s	12.64s	7.1s	11.8s
Made an Error?	1	0	0	0	0	0
	Second Run					
Time Taken	8.7s	12.4s	17.4s	10.49s	8.9s	9.8s
Made an Error?	1	0	0	0	1	0
	Feedback (Rating Out of 5)					
Attractiveness	2	3	4	1	3	4
Ease of Use	2	3	2	2	2	4
Needs Redesign	5	3	4	5	3	5
Grid Layout (GL)						
	First Run					
Time Taken	7.2s	10.8s	12.1s	18.09s	5.4s	10.5s
Made an Error?	1	0	1	1	0	1
	Second Run					
Time Taken	7.9s	9.4s	9.9s	13.25s	5.9s	9.7s
Made an Error?	1	0	0	0	0	0
	Feedback (Rating Out of 5)					
Attractiveness	5	4	5	3	4	5
Ease of Use	5	4	5	3	5	5
Needs Redesign	1	1	2	3	1	1

Unaligned Layout (UL)						
	First Run					
Time Taken	8.3s	10.2s	17.9s	8.72s	7.5s	11.2s
Made an Error?	1	0	1	1	1	1
	Second Run					
Time Taken	6.9s	13.0s	9.0s	10.61s	6.8s	8.5s
Made an Error?	0	1	0	0	1	0
	Feedback (Rating Out of 5)					
Attractiveness	3	2	1	2	2	3
Ease of Use	3	1	1	3	1	4
Needs Redesign	4	5	5	4	4	2
Layout Preference – Best to Worst						
First	GL	FTL	GL	FTL	FTL	FTL
Second	FTL	GL	FTL	UL	GL	GL
Third	CGSL	CGSL	CGSL	GL	CGSL	UL
Fourth	UL	UL	UL	CGSL	UL	CGSL

Appendix F – White-Box Test Plan

Test	Input	Process	Actual Output	Expected Output	Check
Rearrange 4 windows, automatically add to groups	Press Rearrange All button	AutoArrange( )	50% + 23:77 layout applied, 1 category created	50% + 23:77 layout applied, 1 category created	**PASS**
Rearrange 6 windows, automatically add to groups	Press Rearrange All button	AutoArrange( )	50 + 23:77 layout applied to Category 1, Category 2 docked left and right	50 + 23:77 layout applied to Category 1, Category 2 docked left and right, hide Category 1	**PASS**
Rearrange 2 windows to existing arrangement, Category 1 & 2 both have 1 free space	Press Rearrange All button	AutoArrange( )	Add 1 window to Category 1, add 1 window to Category 2, rearrange both	Add 1 window to Category 1, add 1 window to Category 2, rearrange both, hide category 1	**PASS**
Rearrange window from current group	Press Rearrange button	RearrangeCategory(catIndex)	50% + 23:77 layout applied to category	50% + 23:77 layout applied to category	**PASS**
Add window to non-existent Category 8	Hotkey Ctrl + Win + 8	AddtoGroup (activeWindow, 7)	Create Category 8, add active window to Category 8	Create Category 8, add active window to Category 8	**PASS**
Add window to existing Category 8	Hotkey Ctrl + Win + 8	AddtoGroup (activeWindow, 7)	Category 8 window list shows window	Category 8 window list shows window	**PASS**
List windows of Category 8	Press Category #8 button	RefreshTable(7)	Shows added windows, images, names	Shows added windows, images, names	**PASS**
Rename Category 8 to “Specials”	Rename button at Category #8, type name, press Enter	windowgroups [7].CategoryName = “Specials”	Button with Category #8 text changed to Specials	Button with Category #8 text changed to Specials	**PASS**
Delete “Specials”	Press Delete button beside “Specials”	windowgroups [7] = null, RefreshTable(7)	Remove “Specials” controls, interface shows no windows	Remove “Specials” controls, interface shows no windows	**PASS**

Update Category 1 after closing all 4 windows from Category 1	Close all windows from Category 1, press Category 1 button to list windows	RefreshTable(0)	Two windows still remained in the interface with empty window snapshots	Interface shows no windows	**FAIL**
Close all windows	Press Close All button	CloseAll Windows( )	All windows closed except the interface	All windows closed except the interface	**PASS**
Close windows from Category 2 which has random windows with a Notepad with unsaved progress	Hotkey Ctrl + Alt + 2, confirm	CloseCat(1)	Notepad asks to save progress, Category 2 window closed, Category 2 list shows no windows	Notepad asks to save progress, Category 2 windows closed, Category 2 list shows no windows	**PASS**
Kill all windows	Press Kill All button	KillAll Windows( )	All windows closed except the interface	All windows closed except the interface	**PASS**
Kill windows from Category 5 which has random windows with a Notepad with unsaved progress	Hotkey Ctrl + Alt + Shift + 5, confirm	KillCat(4)	Category 5 windows terminated, Category 5 list shows no windows	Category 5 windows terminated, Category 5 list shows no windows	**PASS**
Make selected window the main window of current category	Click on Make Active Window button of desired window	windowGroups [catIndex].Main Window = newMain, RearrangeCateg ory(catIndex)	Selected window becomes main window, category arranged correctly	Selected window becomes main window, category arranged correctly	**PASS**
Swap active window with main window of its category	Hotkey Win + S	windowGroups [catIndex].Main Window = newMain, RearrangeCateg ory(catIndex)	Active window becomes main window instead, category arranged correctly	Active window becomes main window instead, category arranged correctly	**PASS**

Swap active window which is not in any category with main window	Hotkey Win + S	windowGroups [catIndex].Main Window = newMain, RearrangeCateg ory(catIndex)	Error: The selected window does not exist in any category	Error: The selected window does not exist in any category	**PASS**
Turn off AutoDock mode	Hotkey Win + Shift + Tab	Autodock = false	Switch from on to off, show message	Switch from on to off, show message	**PASS**
Turn on AutoDock mode	Hotkey Win + Shift + Tab	Autodock = true	Switch from off to on, show message	Switch from off to on, show message	**PASS**
Dock first window left while in AutoDock	Left Arrow Key	SetWindowPos	Window takes half screen width, full screen height, docked left side	Window takes half screen width, full screen height, docked left side	**PASS**
Dock second window left while in AutoDock	Left Arrow Key	SetWindowPos	First and second window resized to half screen width, half screen height, docked left side. First on top, second below	First and second window resized to half screen width, half screen height, docked left side. First on top, second below	**PASS**
Dock third window left while in AutoDock	Left Arrow Key	SetWindowPos	Left side has 3 windows equally sized	Left side has 3 windows equally sized	**PASS**
Dock a window right while in AutoDock	Right Arrow Key	SetWindowPos	Window takes half screen width, full screen height, docked right side	Window takes half screen width, full screen height, docked right side	**PASS**
Dock a window middle while in AutoDock	Up/Down Arrow Key	SetWindowPos	Left and right side windows resized 1/3 of screen width while retaining arrangement, current window filled in middle	Left and right side windows resized 1/3 of screen width while retaining arrangement, current window filled in middle	**PASS**

Dock window to left while in AutoDock, taskbar set in Left position	Left Arrow Key	SetWindowPos	Window docked left after the taskbar	Window docked left behind the taskbar	**Needs offset**
Dock window while AutoDock OFF	Arrow Keys	-	Nothing happens	Nothing happens	**PASS**
Hide all windows	Hotkey Win + H	HideAll Windows( )	All windows hidden	All windows hidden	**PASS**
Hide windows in Category 1	Hotkey Alt + Shift + 1	HideCat(0)	Windows in Category #1 hidden	Windows in Category #1 hidden	**PASS**
Close windows in Category 1 which are hidden	Press Close Category button	CloseCat(0)	Windows in Category #1 closed, interface still shows half of windows	Windows in Category #1 closed, interface shows no windows for Category #1	**Same as** **Update prob.**
Show all windows	Hotkey Win + G	ShowAll Windows( )	All windows shown except previously hidden windows	All windows shown	**Fix hide logic**
Show windows in Category 2	Hotkey Alt + 2	ShowCat(1)	Hidden windows not shown	Windows in Category #2 shown	**Fix hide logic**
Move current window	Hotkey Shift + Arrow Keys	MoveActive Window(x)	Window moves towards the direction pressed	Window moves towards the direction pressed	**PASS**
Resize current window	Hotkey Ctrl + Shift + Up / Down	ResizeActiveWindow(x)	Window resizes larger if Up pressed, smaller if Down pressed	Window resizes larger if Up pressed, smaller if Down pressed	**PASS**
Rename category	Press Rename button	RenameCat (name, index)	Textbox appears, type name, Enter. Name changes.	Textbox appears, type name, Enter. Name changes.	**PASS**
Starts up when Windows starts	Check the startup box	RegisterKey	Automatically starts application when Windows loads	Automatically starts application when Windows loads	**PASS**

Appendix G – Project Proposal

INVESTIGATION OF HUMAN-COMPUTER INTERACTION USING ALGORITHMIC APPLICATION AS A SOLUTION FOR THE WINDOW MANAGEMENT LIMITATIONS FOUND IN DESKTOP ENVIRONMENTS WITH WINDOWS 7 AS AN EXAMPLE

NICHOLAS W. W. H.

BSc. Computing (Hons.) [3+0]
UOG ID: XXXXXXXXX

1.0 Project Overview

The purpose of this project is to improve the user's control over windows on desktop environments such as Aero in Windows 7. An application will be implemented to run in the background and provide **hotkey** functionality based on the Windows API for users to **manipulate** and **organize** their **windows** on the **screen** with ease.

The main function of the application is to calculate the position of the windows using **algorithms** and resize them to fit to the screen without obstructing each other using hotkeys. The algorithm must account for readability when determining the position and size of windows. This allows the user to quickly keep track of all windows on the screen in the most optimal manner without having to manage them one-by-one, allowing them to **multi-task** effectively and keep track of all windows at once without effort. The user will also be able to move and resize windows using hotkeys which were not originally provided by the desktop environment to control individual or all windows. The application may contain a secondary function to transform windows into small picture snapshots which will be presented in an overlay. This will act as a virtual space for managing windows without clutter as a solution for small screens.

Start Date: 25th September 2013 **End Date:** 24th April 2014

Keywords: hotkey, manipulate, organize, windows, screen, algorithms, multi-task

2.0 Aim

To study the effectiveness of multi-tasking and human-computer interaction using a background application making use of hotkeys and algorithms to easily manipulate windows on a desktop environment.

3.0 Objectives

3.1 Key Research Areas

3.1.1 Identify problems regarding the limitations of window management to users.

3.1.2 Research on HCI usability theories and layouts suitable for desktop environments.

3.1.3 Research on HCI regarding the practical usability of hotkeys and its efficiency.

3.1.4 Perform questionnaires to investigate user preferences on screen layouts at home and office environments.

3.1.5 Research on implementation methods of Windows API for window manipulation.

3.1.6 Analyze user interaction and behaviour on window manipulation.

3.1.7 Analyze Windows API usage and HCI factors for prototype and algorithm design.

3.2 Development

3.2.1 Requirements Planning

3.2.1.1 Develop and finalize project management plans.

3.2.1.2 Perform research on the key research areas regarding HCI and gather high-level requirements based on results.

3.2.1.3 Produce behavioural UML use case diagram on user interaction.

3.2.2 Prototype & Algorithm Design

3.2.2.1 Design the prototype's interface and produce screen layouts.

3.2.2.2 Design a flowchart of processes for the prototype's usage of the algorithm.

3.2.2.3 Design an algorithm to manipulate windows according to user and HCI data.

3.2.3 Construction

3.2.3.1 Revise and develop a working prototype.

3.2.3.2 Test the prototype involving users and gather statistical user input and HCI data.

3.2.4 Implementation

3.2.4.1 Final testing and integration of prototype as a working application.

3.2.4.2 Full documentation and presentation of the project.

4.0 Functional Requirements

4.1 Primary Functions

- Rearrange all windows on screen with a hotkey combination.
- Move and resize active window with hotkeys.
- Automatically dock multiple windows to a portion of a screen with hotkeys by resizing the windows as they are being docked.

4.2 Secondary Functions

- Turn all windows into snapshots in an overlay.
- Group windows into categories.
- Provide an interface for all window groups with snapshots and controls.
- Rearrange, close, show and hide windows by group.
- Rename window groups.

5.0 Non-Functional Requirements

5.1 Readability

The algorithm which resizes and positions the windows must account for readability. If the window is too small, it should take appropriate measures such as minimizing inactive windows.

5.2 Accessibility

It must use easy-to-reach hotkeys and not interfere with the original Windows hotkey functionality. The application will provide ease-of-use with some options such as the choice of running the application at startup and the rebinding of hotkeys.

6.0 Legal, Social, Ethical and Professional Concerns

This project will only involve the development of a standalone background application which integrates to the desktop environment without modifying Windows 7 and its components; therefore, it should not violate any proprietary license or copyright laws.

7.0 Planning (see appendix A)

The approach for this project will follow the Rapid Application Development methodology to gather the requirements dynamically while highly focused on evolutionary prototyping and testing due to high user involvement when creating algorithms for human-computer interaction.

Due to the fixed deadline, the project will be controlled using timeboxing together with the MoSCoW method depending on the deliverables. The project may encounter vague complications such as the inability to acquire solid information during research and tests for user-friendliness, inability to solve algorithmic problems, and commitment to other classes in the degree course which may extend the duration of a work process. These risks will be managed by reducing the project scope through the MoSCoW method to fit into the timebox allocated.

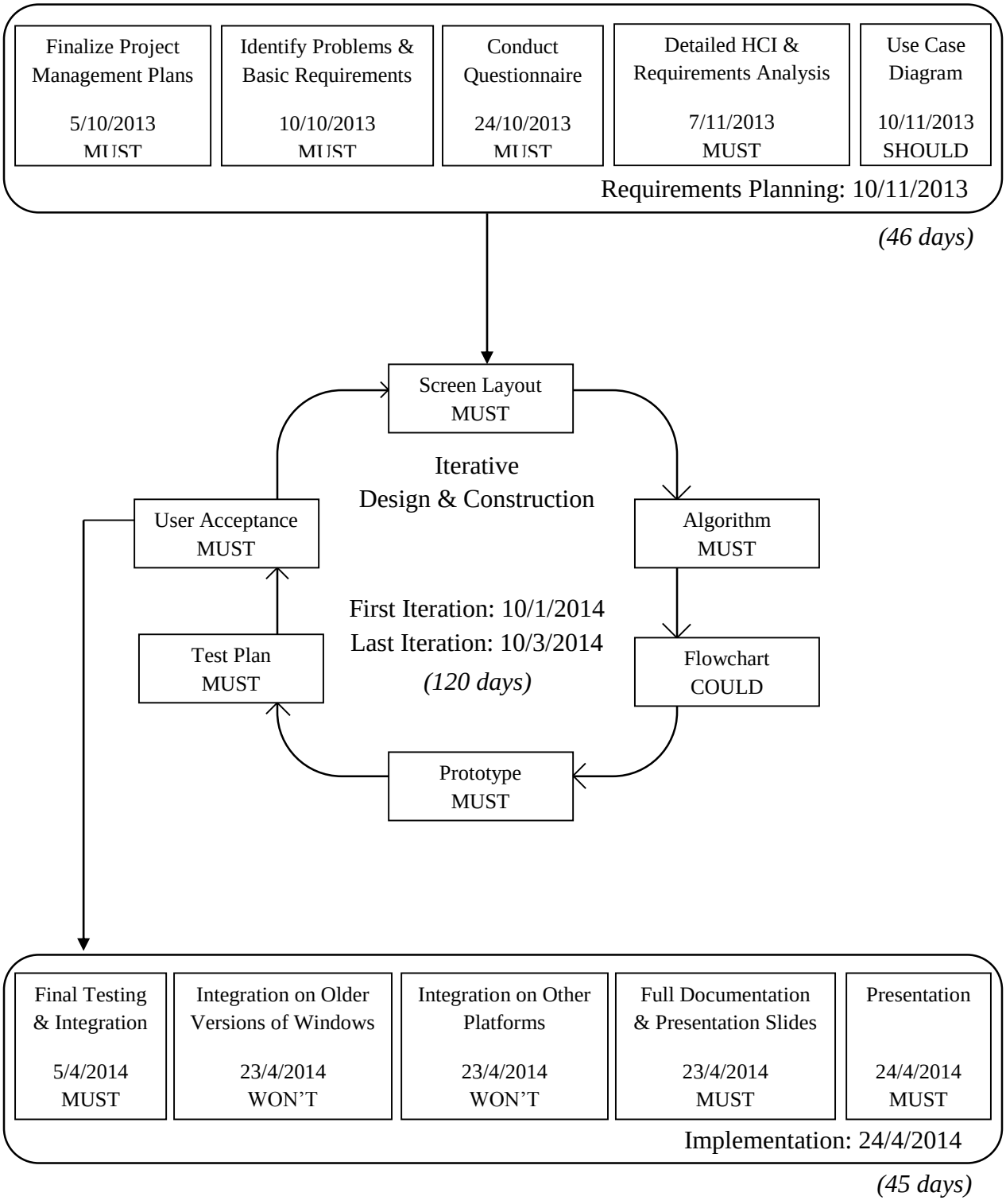
8.0 Initial References

Du Toit, C. (2013). *Rapid Application Development Techniques*. [PowerPoint slides]. Presented for Week 4 of the XXXXXXXX Subject in University of Greenwich.

Microsoft Developer Network. (2013). *Window Functions (Windows)*. [online] Available at: http://msdn.microsoft.com/en-us/library/ff468919(v=vs.85).aspx [Accessed: 23 Sep 2013].

Signatures Redacted

9.0 Proposal Appendix – RAD Model with Timebox Plan



Appendix H – Developed Application Screenshots

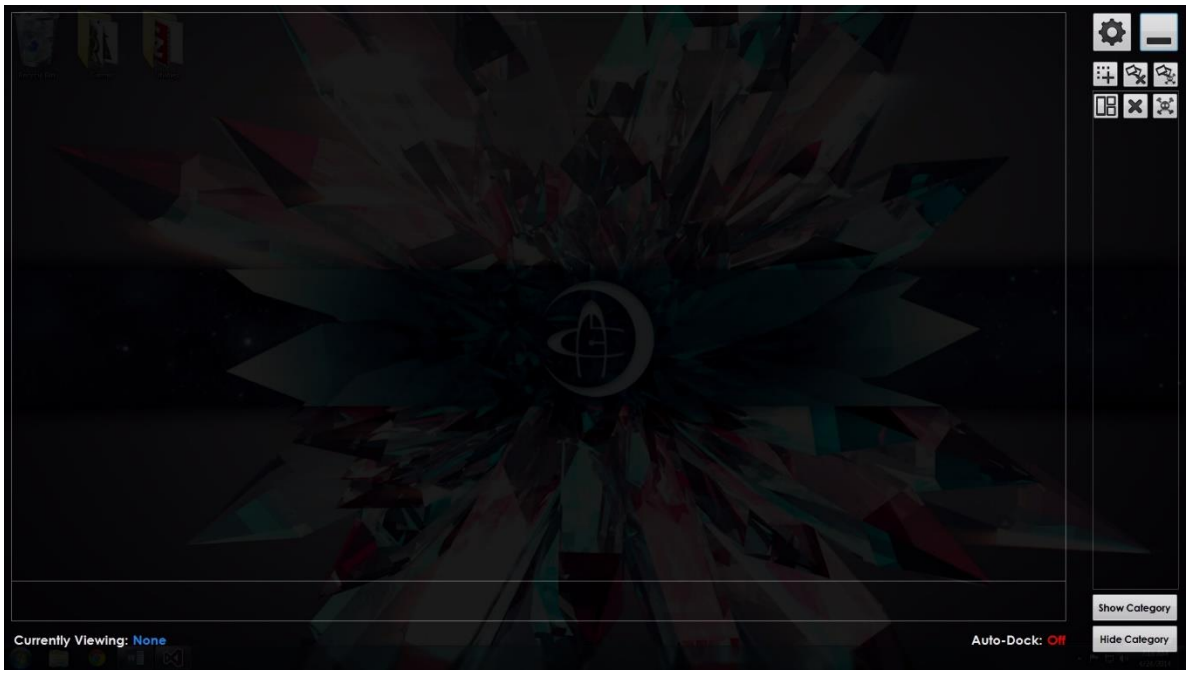


Figure H1: Main Screen

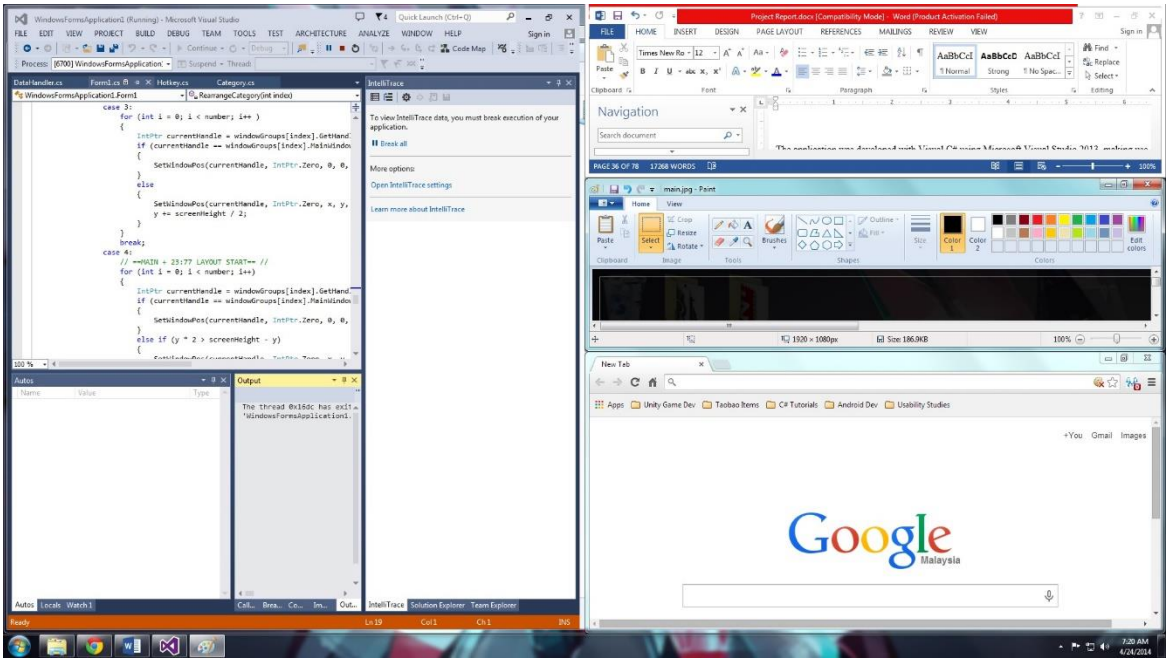


Figure H2: Implemented 50% + 23:77 Layout

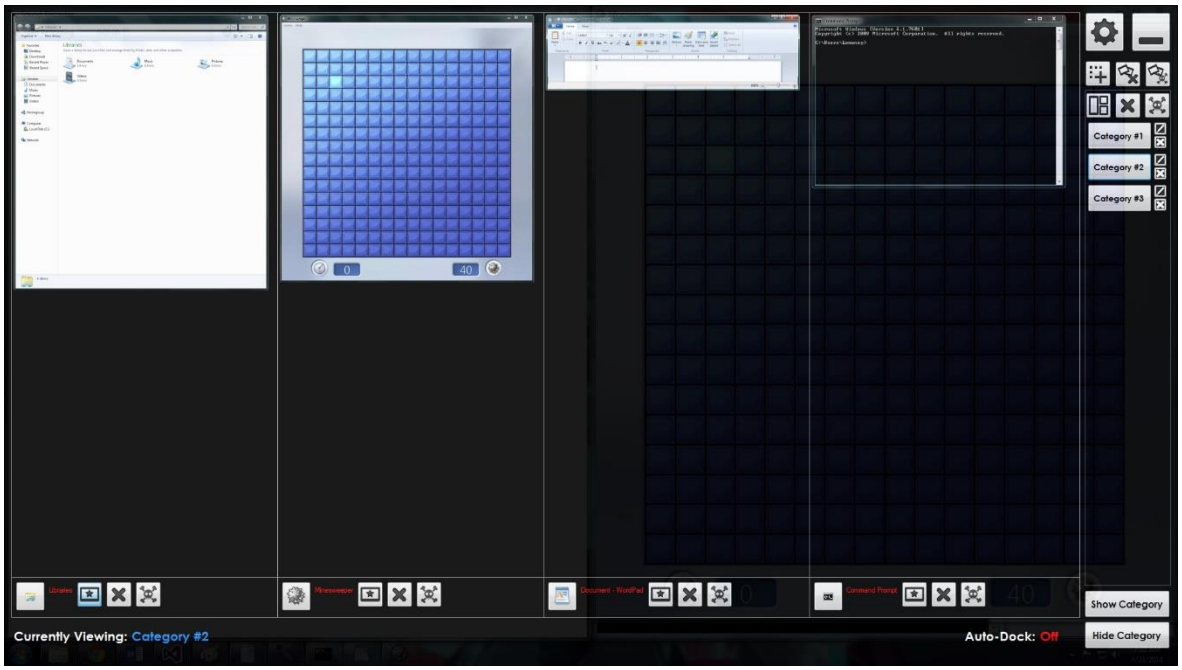


Figure H3: List of Windows & Categories in Interface

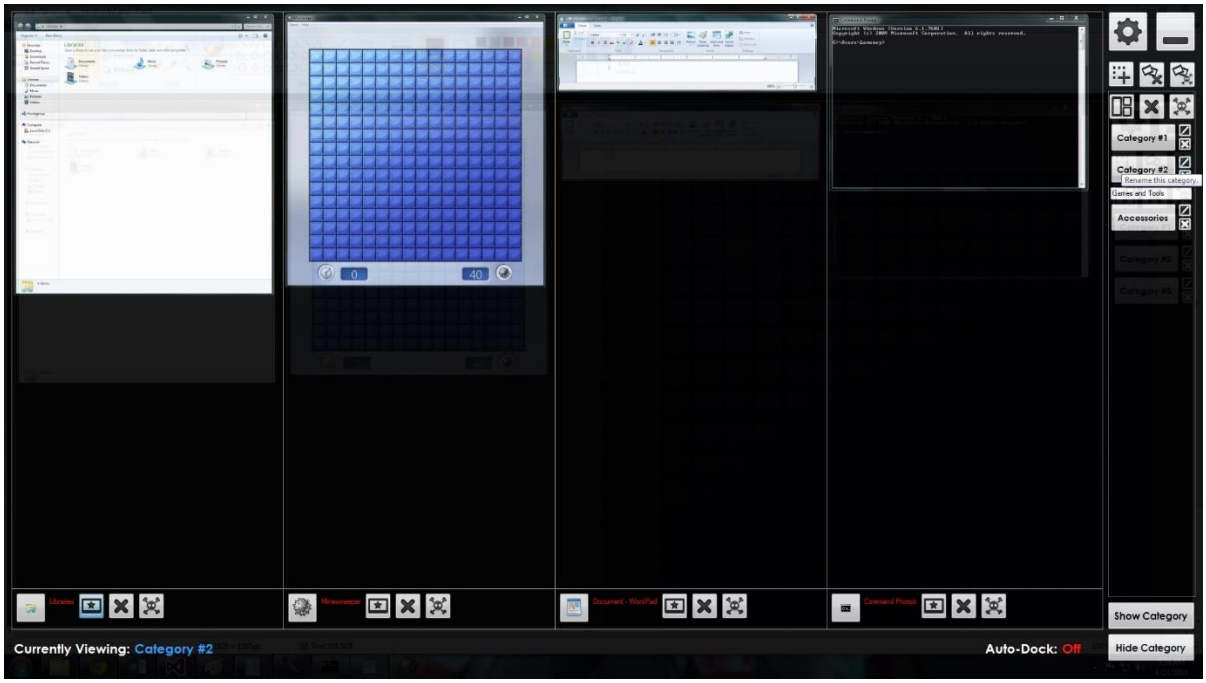


Figure H4: Category Controls & Renaming

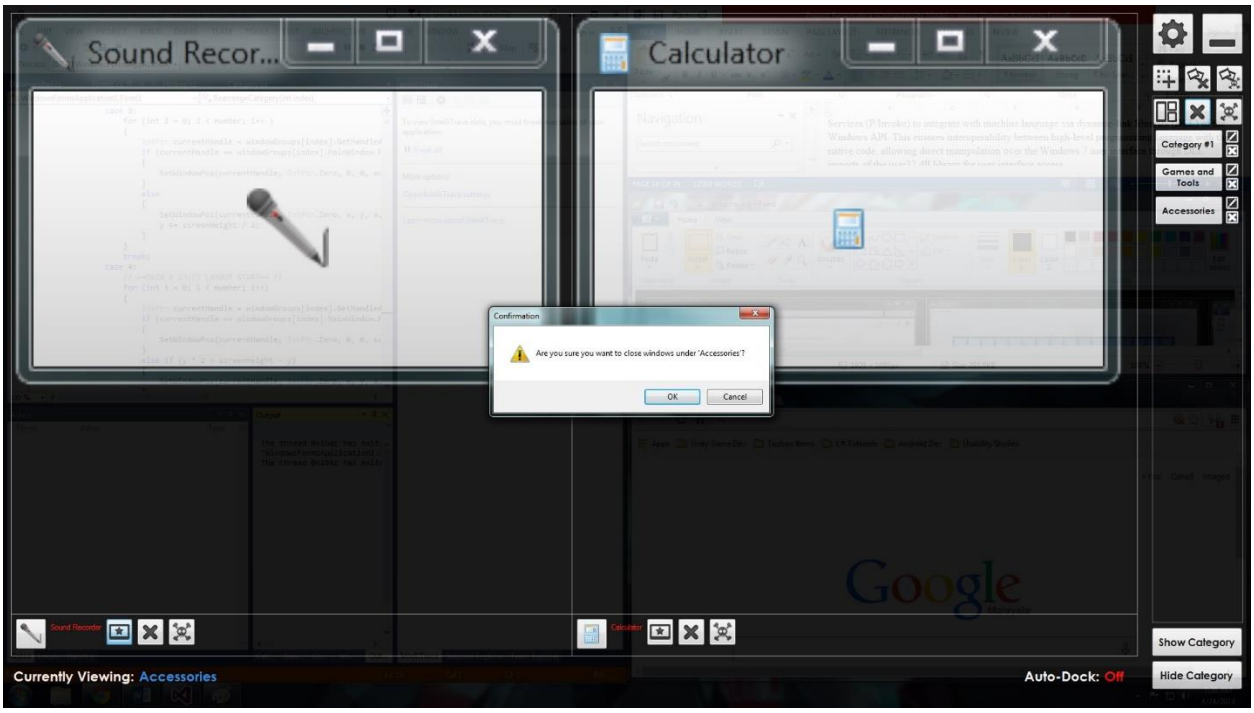


Figure H5: Closing Groups of Windows with Confirmation

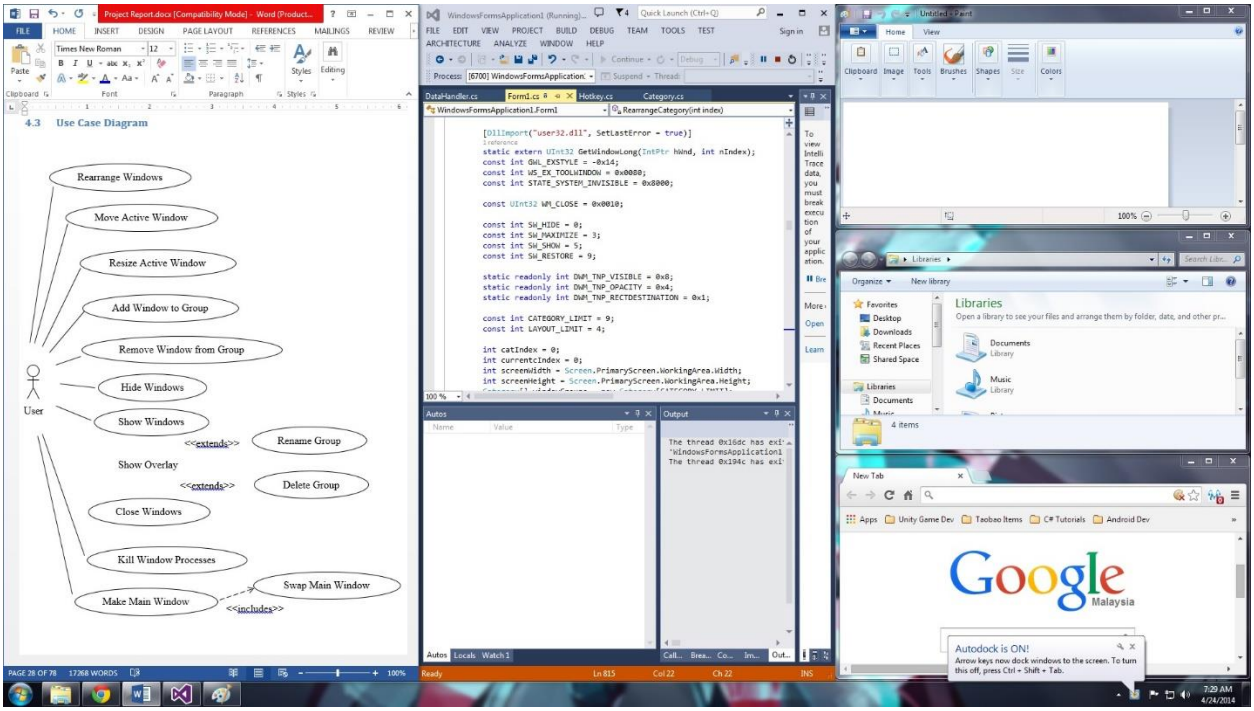


Figure H6: Auto-docking Windows in 3 Divisions

Appendix I – Developed Application Source Code

Hotkey.cs Class

```
// Credits to Curtis Rutland for this Hotkey class from a tutorial forum post.

// Rutland, C. (2010). Global Hotkeys - C# Tutorials. [online] Dream.In.Code.
// Available at: http://www.dreamincode.net/forums/topic/180436-global-hotkeys/
// [Accessed 3 Mar 2014].

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Runtime.InteropServices;


namespace WindowsFormsApplication1
{
    class Hotkey
    {
        [DllImport("user32.dll")]
        private static extern bool RegisterHotKey(IntPtr hWnd, int id, int fsModifiers,
int vk);

        [DllImport("user32.dll")]
        private static extern bool UnregisterHotKey(IntPtr hWnd, int id);

        public int modifier { get; set; }
        public Keys key { get; set; }
        private IntPtr hWnd;
        private int id;

        public Hotkey(int modifier, Keys key, Form form)
        {
            this.modifier = modifier;
            this.key = key;
            this.hWnd = form.Handle;
            id = this.GetHashCode();
        }

        public override int GetHashCode()
        {
            // Generates a unique ID for the hotkey
            return modifier ^ (int)key ^ hWnd.ToInt32();
        }

        public bool Register()
        {
            // Register as global hotkey
            return RegisterHotKey(hWnd, id, modifier, (int)key);
```

```
        }

        public bool Unregister()
        {
            // Unregister
            return UnregisterHotKey(hWnd, id);
        }
    }

     public static class HotkeyConstants
    {
        //modifiers
        public const int NOMOD = 0x0000;
        public const int ALT = 0x0001;
        public const int CTRL = 0x0002;
        public const int SHIFT = 0x0004;
        public const int WIN = 0x0008;

        //windows message id for hotkey
        public const int WM_HOTKEY_MSG_ID = 0x0312;
    }
}
```

Window.cs Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WindowsFormsApplication1
{
    class Window
    {
        public IntPtr Handle { get; set; }
        public int xPos { get; set; }
        public int yPos { get; set; }
        public int Width { get; set; }
        public int Height { get; set; }
        public string Title { get; set; }

        public Window(IntPtr hWnd, string t, int x, int y, int w, int h)
        {
            Handle = hWnd;
            Title = t;
            xPos = x;
            yPos = y;
            Width = w;
            Height = h;
        }
    }
}
```

Category.cs Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WindowsFormsApplication1
{
    class Category
    {
        public string CategoryName { get; set; }
        public Window MainWindow { get; set; }

        private List<Window> Windows = new List<Window>();

        public Category(string name, Window window, int capacity)
        {
            CategoryName = name;
            MainWindow = window;
            Windows.Capacity = capacity;
        }

        public Category(string name)
        {
            CategoryName = name;
        }

        public Category(string name, int capacity)
        {
            CategoryName = name;
            Windows.Capacity = capacity;
        }

        public bool AddWindow(Window window)
        {
            if (isFull() || hasWindow(window))
            {
                return false;
            }
            else
            {
                Windows.Add(window);
                return true;
            }
        }

        public bool AddtoFirst(Window window)
        {
            if (isFull() || hasWindow(window))
            {
                return false;
            }
```

```
        else
        {
            Windows.Reverse();
            Windows.Add(window);
            Windows.Reverse();
            return true;
        }
    }

    public void RemoveWindow(Window window)
    {
        //Windows.Remove(window);
        Windows.RemoveAll(w => w.Handle == window.Handle);
        if (isMainWindow(window)) { SetDefaultMainWindow(); }
    }

    public void RemoveAt(int index)
    {
        Window temp = Windows.ElementAt(index);
        Windows.RemoveAt(index);
        if (isMainWindow(temp)) { SetDefaultMainWindow(); }
    }

    public void SetDefaultMainWindow()
    {
        if (!isEmpty())
        {
            MainWindow = Windows.ElementAt(0);
        }
    }

    public void Clear()
    {
        Windows.Clear();
    }

    public bool isFull()
    {
        if (Windows.Count == Windows.Capacity) { return true; }
        else { return false; }
    }

    public bool isEmpty()
    {
        if (!Windows.Any()) { return true; } else { return false; }
    }

    public bool isMainWindow(Window window)
    {
        if (MainWindow == null) { return false; }
        if (window.Handle == MainWindow.Handle) { return true; }
        return false;
    }
```

```
        public bool hasWindow(Window window)
        {
            for (int i = 0; i < Windows.Count; i++)
            {
                if (Windows[i].Handle == window.Handle) { return true; }
            }
            return false;
        }

        public int numberofWindows()
        {
            return Windows.Count;
        }

        public Window GetWindowAt(int x)
        {
            return Windows.ElementAt(x);
        }


        public IntPtr GetHandleAt(int x)
        {
            return Windows.ElementAt(x).Handle;
        }

        public void SetCapacity(int c)
        {
            Windows.Capacity = c;
        }
    }
}
```

DataHandler.cs Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;

namespace WindowsFormsApplication1
{
    class DataHandler
    {
        private XDocument Document;

        public DataHandler(XDocument doc)
        {
            Document = doc;
        }

        public static void InitializeData() {
            XDocument hotkeyData = new XDocument(
                    new XDeclaration("1.0","utf8","yes"),
                    new XComment("Keybinds for Window Management Tool"),

                    new XElement("Hotkeys",
                        new XElement("MoveUp",
                            new XElement("Modifier", HotkeyConstants.SHIFT.ToString()),
                            new XElement("Key", Keys.Up.ToString())
                            ),
                        new XElement("MoveDown",
                            new XElement("Modifier", HotkeyConstants.SHIFT.ToString()),
                            new XElement("Key", Keys.Down.ToString())
                            ),
                        new XElement("MoveLeft",
                            new XElement("Modifier", HotkeyConstants.SHIFT.ToString()),
                            new XElement("Key", Keys.Left.ToString())
                            ),
                        new XElement("MoveRight",
                            new XElement("Modifier", HotkeyConstants.SHIFT.ToString()),
                            new XElement("Key", Keys.Right.ToString())
                            ),
                        new XElement("SizeUp",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.Up.ToString())
                            ),
                        new XElement("SizeDown",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.Down.ToString())
                            ),
                        new XElement("ArrangeAll",
                            new XElement("Modifier", (HotkeyConstants.WIN).ToString()),
```

```
                            new XElement("Key", Keys.A.ToString())
                            ),
                        new XElement("ArrangeCategory1",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D1.ToString())
                            ),
                        new XElement("ArrangeCategory2",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D2.ToString())
                            ),
                        new XElement("ArrangeCategory3",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D3.ToString())
                            ),
                        new XElement("ArrangeCategory4",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D4.ToString())
                            ),
                        new XElement("ArrangeCategory5",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D5.ToString())
                            ),
                        new XElement("ArrangeCategory6",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D6.ToString())
                            ),
                        new XElement("ArrangeCategory7",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D7.ToString())
                            ),
                        new XElement("ArrangeCategory8",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D8.ToString())
                            ),
                        new XElement("ArrangeCategory9",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D9.ToString())
                            ),
                        new XElement("CloseAll",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.End.ToString())
                            ),
                        new XElement("CloseCategory1",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D1.ToString())
```

```
                            ),
                        new XElement("CloseCategory2",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D2.ToString())
                            ),
                        new XElement("CloseCategory3",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D3.ToString())
                            ),
                        new XElement("CloseCategory4",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D4.ToString())
                            ),
                        new XElement("CloseCategory5",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D5.ToString())
                            ),
                        new XElement("CloseCategory6",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D6.ToString())
                            ),
                        new XElement("CloseCategory7",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D7.ToString())
                            ),
                        new XElement("CloseCategory8",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D8.ToString())
                            ),
                        new XElement("CloseCategory9",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D9.ToString())
                            ),
                        new XElement("KillAll",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.End.ToString())
                            ),
                        new XElement("KillCategory1",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D1.ToString())
                            ),
                        new XElement("KillCategory2",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D2.ToString())
                            ),
```

```
                        new XElement("KillCategory3",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D3.ToString())
                            ),
                        new XElement("KillCategory4",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D4.ToString())
                            ),
                        new XElement("KillCategory5",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D5.ToString())
                            ),
                        new XElement("KillCategory6",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D6.ToString())
                            ),
                        new XElement("KillCategory7",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D7.ToString())
                            ),
                        new XElement("KillCategory8",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D8.ToString())
                            ),
                        new XElement("KillCategory9",
                            new XElement("Modifier", (HotkeyConstants.CTRL +
HotkeyConstants.ALT + HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D9.ToString())
                            ),
                        new XElement("AddToCategory1",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D1.ToString())
                            ),
                        new XElement("AddToCategory2",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D2.ToString())
                            ),
                        new XElement("AddToCategory3",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D3.ToString())
                            ),
                        new XElement("AddToCategory4",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D4.ToString())
                            ),
                        new XElement("AddToCategory5",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D5.ToString())
                            ),
                        new XElement("AddToCategory6",
```

```
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D6.ToString())
                            ),
                        new XElement("AddToCategory7",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D7.ToString())
                            ),
                        new XElement("AddToCategory8",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D8.ToString())
                            ),
                        new XElement("AddToCategory9",
                            new XElement("Modifier", (HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.D9.ToString())
                            ),
                        new XElement("AutodockToggle",
                            new XElement("Modifier", (HotkeyConstants.SHIFT +
HotkeyConstants.CTRL).ToString()),
                            new XElement("Key", Keys.Tab.ToString())
                            ),
                        new XElement("DockUp",
                            new XElement("Modifier", (HotkeyConstants.NOMOD).ToString()),
                            new XElement("Key", Keys.Up.ToString())
                            ),
                        new XElement("DockDown",
                            new XElement("Modifier", (HotkeyConstants.NOMOD).ToString()),
                            new XElement("Key", Keys.Down.ToString())
                            ),
                        new XElement("DockLeft",
                            new XElement("Modifier", (HotkeyConstants.NOMOD).ToString()),
                            new XElement("Key", Keys.Left.ToString())
                            ),
                        new XElement("DockRight",
                            new XElement("Modifier", (HotkeyConstants.NOMOD).ToString()),
                            new XElement("Key", Keys.Right.ToString())
                            ),
                        new XElement("DockAlt",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString())
                            ),
                        new XElement("ShowAll",
                            new XElement("Modifier", (HotkeyConstants.WIN).ToString()),
                            new XElement("Key", Keys.G.ToString())
                            ),
                        new XElement("ShowCategory1",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D1.ToString())
                            ),
                        new XElement("ShowCategory2",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D2.ToString())
                            ),
                        new XElement("ShowCategory3",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D3.ToString())
                            ),
                        new XElement("ShowCategory4",
```

```
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D4.ToString())
                            ),
                        new XElement("ShowCategory5",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D5.ToString())
                            ),
                        new XElement("ShowCategory6",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D6.ToString())
                            ),
                        new XElement("ShowCategory7",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D7.ToString())
                            ),
                        new XElement("ShowCategory8",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D8.ToString())
                            ),
                        new XElement("ShowCategory9",
                            new XElement("Modifier", (HotkeyConstants.ALT).ToString()),
                            new XElement("Key", Keys.D9.ToString())
                            ),
                        new XElement("HideAll",
                            new XElement("Modifier", (HotkeyConstants.WIN).ToString()),
                            new XElement("Key", Keys.H.ToString())
                            ),
                        new XElement("HideCategory1",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D1.ToString())
                            ),
                        new XElement("HideCategory2",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D2.ToString())
                            ),
                        new XElement("HideCategory3",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D3.ToString())
                            ),
                        new XElement("HideCategory4",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D4.ToString())
                            ),
                        new XElement("HideCategory5",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D5.ToString())
                            ),
                        new XElement("HideCategory6",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D6.ToString())
```

```
                            ),
                        new XElement("HideCategory7",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D7.ToString())
                            ),
                        new XElement("HideCategory8",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D8.ToString())
                            ),
                        new XElement("HideCategory9",
                            new XElement("Modifier", (HotkeyConstants.ALT +
HotkeyConstants.SHIFT).ToString()),
                            new XElement("Key", Keys.D9.ToString())
                            ),
                        new XElement("Restore",
                            new XElement("Modifier", (HotkeyConstants.WIN).ToString()),
                            new XElement("Key", Keys.W.ToString())
                            ),
                        new XElement("SwapMainWindow",
                            new XElement("Modifier", (HotkeyConstants.WIN).ToString()),
                            new XElement("Key", Keys.S.ToString())
                            )
                ));
            hotkeyData.Save("hotkeys.xml");
        }



        public Keys GetKey(string element)
        {
            var query = Document.Descendants(element).Select(s => new
            {
                Key = s.Element("Key").Value
            }).FirstOrDefault();
            string keyString = query.Key;
            return (Keys)Enum.Parse(typeof(Keys), keyString);
        }


        public int GetMod(string element)
        {
            var query = Document.Descendants(element).Select(s => new
            {
                Modifier = s.Element("Modifier").Value
            }).FirstOrDefault();
            string modString = query.Modifier;

            return int.Parse(modString);
        }
    }
}
```

Form1.cs (Main Controller) Class

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        // WINDOW MANAGEMENT TOOL 2014
        // Coded by: Nicholas W. W. H (RailKill)
        // This application is created as part of a disseration for
        // University of Greenwich's BSc. Computing programme.

        // DO NOT DISTRIBUTE WITHOUT PERMISSION

        // CREDITS:
        // Credits to Paul Dinham, Curtis Rutland, Bart De Smet and Jani Hartikainen for
        // providing online tutorials and solutions which were adapted and used in the
        // development of this application. Full credits were given and cited on their
        // corresponding code sections.

        // SPECIAL THANKS TO EVAN BROOKS
        // A deviantART user named 'brsev' for creating the Token icon set which was used
        // in this application, free for personal use under the Creative Commons License.
        //
        // Brooks, E. (2009). Token by brsev on deviantART.
        // Available at: http://brsev.deviantart.com/art/Token-128429570
        // [Accessed: 21 Jan 2014].

        // NOTES:
        // Prepare for messy code, no time to clean up, too busy developing.
        // Some functions belong in separate classes, do it next time, for reusability.


        [StructLayout(LayoutKind.Sequential)]
        public struct RECT
        {
            public int Left, Top, Right, Bottom;

            public RECT(int left, int top, int right, int bottom)
            {
                Left = left;
                Top = top;
                Right = right;
```

```
            Bottom = bottom;
        }

        public RECT(System.Drawing.Rectangle r) : this(r.Left, r.Top, r.Right,
r.Bottom) { }

        public int X
        {
            get { return Left; }
            set { Right -= (Left - value); Left = value; }
        }

        public int Y
        {
            get { return Top; }
            set { Bottom -= (Top - value); Top = value; }
        }

        public int Height
        {
            get { return Bottom - Top; }
            set { Bottom = value + Top; }
        }

        public int Width
        {
            get { return Right - Left; }
            set { Right = value + Left; }
        }

        public System.Drawing.Point Location
        {
            get { return new System.Drawing.Point(Left, Top); }
            set { X = value.X; Y = value.Y; }
        }

        public System.Drawing.Size Size
        {
            get { return new System.Drawing.Size(Width, Height); }
            set { Width = value.Width; Height = value.Height; }
        }

        public static implicit operator System.Drawing.Rectangle(RECT r)
        {
            return new System.Drawing.Rectangle(r.Left, r.Top, r.Width, r.Height);
        }

        public static implicit operator RECT(System.Drawing.Rectangle r)
        {
            return new RECT(r);
        }

        public static bool operator ==(RECT r1, RECT r2)
        {
            return r1.Equals(r2);
        }
```

```
            public static bool operator !=(RECT r1, RECT r2)
            {
                return !r1.Equals(r2);
            }

            public bool Equals(RECT r)
            {
                return r.Left == Left && r.Top == Top && r.Right == Right && r.Bottom ==
Bottom;
            }

            public override bool Equals(object obj)
            {
                if (obj is RECT)
                    return Equals((RECT)obj);
                else if (obj is System.Drawing.Rectangle)
                    return Equals(new RECT((System.Drawing.Rectangle)obj));
                return false;
            }

            public override int GetHashCode()
            {
                return ((System.Drawing.Rectangle)this).GetHashCode();
            }

            public override string ToString()
            {
                return string.Format(System.Globalization.CultureInfo.CurrentCulture,
"{{Left={0},Top={1},Right={2},Bottom={3}}}", Left, Top, Right, Bottom);
            }
        }

        [StructLayout(LayoutKind.Sequential)]
        struct TITLEBARINFO
        {
            public const int CCHILDREN_TITLEBAR = 5;
            public uint cbSize;
            public RECT rcTitleBar;
            [MarshalAs(UnmanagedType.ByValArray, SizeConst = CCHILDREN_TITLEBAR + 1)]
            public uint[] rgstate;
        }

        [DllImport("user32.dll")]
        [return: MarshalAs(UnmanagedType.Bool)]
        static extern bool GetTitleBarInfo(IntPtr hwnd, ref TITLEBARINFO ti);

        [DllImport("user32.dll")]
        [return: MarshalAs(UnmanagedType.Bool)]
        static extern bool SetWindowPos(IntPtr hWnd, IntPtr hWndInsertAfter, int X, int Y,
int cx, int cy, int wFlags);
        const short SWP_NOMOVE = 0X2;
        const short SWP_NOSIZE = 1;
        const short SWP_NOZORDER = 0X4;
        const int SWP_SHOWWINDOW = 0x0040;
```

```
        [DllImport("user32.dll")]
        [return: MarshalAs(UnmanagedType.Bool)]
        static extern bool IsWindowVisible(IntPtr hWnd);

        [DllImport("user32.dll", ExactSpelling = true)]
        static extern IntPtr GetAncestor(IntPtr hwnd, uint gaFlags);
        const uint GA_PARENT = 1;
        const uint GA_ROOT = 2;
        const uint GA_ROOTOWNER = 3;

        [DllImport("user32.dll", SetLastError = true)]
        static extern IntPtr GetWindow(IntPtr hWnd, GetWindow_Cmd uCmd);

        enum GetWindow_Cmd : uint
        {
            GW_HWNDFIRST = 0,
            GW_HWNDLAST = 1,
            GW_HWNDNEXT = 2,
            GW_HWNDPREV = 3,
            GW_OWNER = 4,
            GW_CHILD = 5,
            GW_ENABLEDPOPUP = 6
        }

        [DllImport("user32.dll")]
        static extern IntPtr GetLastActivePopup(IntPtr hWnd);

        [DllImport("user32.dll")]
        private static extern IntPtr GetForegroundWindow();

        [DllImport("user32.dll")]
        [return: MarshalAs(UnmanagedType.Bool)]
        static extern bool SetForegroundWindow(IntPtr hWnd);

        [DllImport("user32.dll")]
        [return: MarshalAs(UnmanagedType.Bool)]
        static extern bool EnumWindows(EnumWindowsProc lpEnumFunc, IntPtr lParam);

        private delegate bool EnumWindowsProc(IntPtr hWnd, IntPtr lParam);

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = true)]
        static extern int GetWindowText(IntPtr hWnd, StringBuilder lpString, int
nMaxCount);

        [DllImport("user32.dll", SetLastError = true, CharSet = CharSet.Auto)]
        static extern int GetWindowTextLength(IntPtr hWnd);

        [DllImport("user32.dll")]
        public static extern bool GetWindowRect(IntPtr hwnd, ref RECT rectangle);

        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        static extern IntPtr SendMessage(IntPtr hWnd, UInt32 Msg, IntPtr wParam, IntPtr
lParam);

        [DllImport("user32.dll", SetLastError = true)]
        static extern uint GetWindowThreadProcessId(IntPtr hWnd, out int processId);
```

```
        [DllImport("dwmapi.dll", SetLastError = true)]
        static extern int DwmRegisterThumbnail(IntPtr dest, IntPtr src, out IntPtr
thumb);

        [DllImport("dwmapi.dll")]
        static extern int DwmUnregisterThumbnail(IntPtr thumb);

        [DllImport("dwmapi.dll")]
        static extern int DwmUpdateThumbnailProperties(IntPtr hThumb, ref
DWM_THUMBNAIL_PROPERTIES props);


        [StructLayout(LayoutKind.Sequential)]
        internal struct DWM_THUMBNAIL_PROPERTIES
        {
            public int dwFlags;
            public RECT rcDestination;
            public RECT rcSource;
            public byte opacity;
            public bool fVisible;
            public bool fSourceClientAreaOnly;
        }

        // GET ICON IMPORTS AND CONSTANTS //
        // This section is imported to be used as part of Hartikainen's (2007) code.
        // See GetIcon function for more.
        public const int GCL_HICONSM = -34;
        public const int GCL_HICON = -14;

        public const int ICON_SMALL = 0;
        public const int ICON_BIG = 1;
        public const int ICON_SMALL2 = 2;

        public const int WM_GETICON = 0x7F;

        public static IntPtr GetClassLongPtr(IntPtr hWnd, int nIndex)
        {
            if (IntPtr.Size > 4)
                return GetClassLongPtr64(hWnd, nIndex);
            else
                return new IntPtr(GetClassLongPtr32(hWnd, nIndex));
        }

        [DllImport("user32.dll", EntryPoint = "GetClassLong")]
        public static extern uint GetClassLongPtr32(IntPtr hWnd, int nIndex);

        [DllImport("user32.dll", EntryPoint = "GetClassLongPtr")]
        public static extern IntPtr GetClassLongPtr64(IntPtr hWnd, int nIndex);

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError = false)]
        static extern IntPtr SendMessage(IntPtr hWnd, int Msg, int wParam, int lParam);
        // ICON IMPORTS AND CONSTANTS END //

        [DllImport("user32.dll")]
        static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
```

```
[DllImport("user32.dll", SetLastError = true)]
static extern UInt32 GetWindowLong(IntPtr hWnd, int nIndex);
const int GWL_EXSTYLE = -0x14;
const int WS_EX_TOOLWINDOW = 0x0080;
const int STATE_SYSTEM_INVISIBLE = 0x8000;

const UInt32 WM_CLOSE = 0x0010;

const int SW_HIDE = 0;
const int SW_MAXIMIZE = 3;
const int SW_SHOW = 5;
const int SW_RESTORE = 9;

static readonly int DWM_TNP_VISIBLE = 0x8;
static readonly int DWM_TNP_OPACITY = 0x4;
static readonly int DWM_TNP_RECTDESTINATION = 0x1;

const int CATEGORY_LIMIT = 9;
const int LAYOUT_LIMIT = 4;

int catIndex = 0;
int currentcIndex = 0;
int screenWidth = Screen.PrimaryScreen.WorkingArea.Width;
int screenHeight = Screen.PrimaryScreen.WorkingArea.Height;
Category[] windowGroups = new Category[CATEGORY_LIMIT];
Category tempGroup = new Category("Temporary");
Category hiddenGroup = new Category("Hidden");
List<RadioButton> activeButtons = new List<RadioButton>();

// General Keys //
private Hotkey autoDockKey, dockUp, dockDown, dockLeft, dockRight,
    dockAltUp, dockAltDown, dockAltLeft, dockAltRight, swapMainKey, restoreKey;
// Move Window Keys //
private Hotkey moveLeftKey, moveDownKey, moveUpKey, moveRightKey;
// Resize Window Keys //
private Hotkey sizeUpKey, sizeDownKey;
// Arrange Keys //
private Hotkey arrangeKey;
private Hotkey[] arrange = new Hotkey[9];
// Add Category Keys //
private Hotkey[] addCat = new Hotkey[9];
// Close Keys //
private Hotkey closeKey;
private Hotkey[] close = new Hotkey[9];
// Kill Keys //
private Hotkey killKey;
private Hotkey[] kill = new Hotkey[9];
// Show Keys //
private Hotkey showKey;
private Hotkey[] show = new Hotkey[9];
// Hide Keys
private Hotkey hideKey;
private Hotkey[] hide = new Hotkey[9];
```

```
        private IntPtr[] preview = { IntPtr.Zero, IntPtr.Zero, IntPtr.Zero,
IntPtr.Zero };
        private bool autoDock = false;
        private Category[] adWindows = new Category[3];
        private Form2 settingsForm = new Form2();
        private DataHandler data;



        public Form1()
        {
            InitializeComponent();
            try
            {
                data = new DataHandler(XDocument.Load("hotkeys.xml"));
            }
            catch (Exception e)
            {
                try
                {
                    DataHandler.InitializeData();
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Error: " + ex + "\n\nUnable to initialize hotkey
data. Try running as Administrator.");
                    System.Environment.Exit(1);
                }
            }
            finally
            {
                if (data == null) { data = new
DataHandler(XDocument.Load("hotkeys.xml")); }
            }

            // Set Hidden Group Capacity
            hiddenGroup.SetCapacity(100);
            tempGroup.SetCapacity(100);

            // Assign General Keys //
            autoDockKey = new Hotkey(data.GetMod("AutodockToggle"),
data.GetKey("AutodockToggle"), this);
            dockUp = new Hotkey(data.GetMod("DockUp"), data.GetKey("DockUp"), this);
            dockDown = new Hotkey(data.GetMod("DockDown"), data.GetKey("DockDown"),
this);
            dockLeft = new Hotkey(data.GetMod("DockLeft"), data.GetKey("DockLeft"),
this);
            dockRight = new Hotkey(data.GetMod("DockRight"), data.GetKey("DockRight"),
this);

            dockAltUp = new Hotkey(data.GetMod("DockAlt"), data.GetKey("DockUp"), this);
            dockAltDown = new Hotkey(data.GetMod("DockAlt"), data.GetKey("DockDown"),
this);
            dockAltLeft = new Hotkey(data.GetMod("DockAlt"), data.GetKey("DockLeft"),
this);
```

```
            dockAltRight = new Hotkey(data.GetMod("DockAlt"), data.GetKey("DockRight"),
this);
            restoreKey = new Hotkey(data.GetMod("Restore"), data.GetKey("Restore"),
this);

            // Assign Move Window Keys //
            moveUpKey = new Hotkey(data.GetMod("MoveUp"), data.GetKey("MoveUp"), this);
            moveDownKey = new Hotkey(data.GetMod("MoveDown"), data.GetKey("MoveDown"),
this);
            moveLeftKey = new Hotkey(data.GetMod("MoveLeft"), data.GetKey("MoveLeft"),
this);
            moveRightKey = new Hotkey(data.GetMod("MoveRight"), data.GetKey("MoveRight"),
this);

            // Assign Size Keys //
            sizeUpKey = new Hotkey(data.GetMod("SizeUp"), data.GetKey("SizeUp"), this);
            sizeDownKey = new Hotkey(data.GetMod("SizeUp"), data.GetKey("SizeDown"),
this);

            // Assign Other Keys //
            arrangeKey = new Hotkey(data.GetMod("ArrangeAll"), data.GetKey("ArrangeAll"),
this);
            closeKey = new Hotkey(data.GetMod("CloseAll"), data.GetKey("CloseAll"),
this);
            killKey = new Hotkey(data.GetMod("KillAll"), data.GetKey("KillAll"), this);
            swapMainKey = new Hotkey(data.GetMod("SwapMainWindow"),
data.GetKey("SwapMainWindow"), this);
            showKey = new Hotkey(data.GetMod("ShowAll"), data.GetKey("ShowAll"), this);
            hideKey = new Hotkey(data.GetMod("HideAll"), data.GetKey("HideAll"), this);

            for (int i = 0; i < CATEGORY_LIMIT; i++)
            {
                int j = i + 1;
                arrange[i] = new Hotkey(data.GetMod("ArrangeCategory" + j),
data.GetKey("ArrangeCategory" + j), this);
                addCat[i] = new Hotkey(data.GetMod("AddToCategory" + j),
data.GetKey("AddToCategory" + j), this);
                close[i] = new Hotkey(data.GetMod("CloseCategory" + j),
data.GetKey("CloseCategory" + j), this);
                kill[i] = new Hotkey(data.GetMod("KillCategory" + j),
data.GetKey("KillCategory" + j), this);
                show[i] = new Hotkey(data.GetMod("ShowCategory" + j),
data.GetKey("ShowCategory" + j), this);
                hide[i] = new Hotkey(data.GetMod("HideCategory" + j),
data.GetKey("HideCategory" + j), this);
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Register all the hotkeys
            autoDockKey.Register();

            restoreKey.Register();

            moveUpKey.Register();
```

```
            moveDownKey.Register();
            moveLeftKey.Register();
            moveRightKey.Register();
            sizeUpKey.Register();
            sizeDownKey.Register();

            arrangeKey.Register();
            closeKey.Register();
            killKey.Register();
            swapMainKey.Register();
            showKey.Register();
            hideKey.Register();

            for (int i = 0; i < CATEGORY_LIMIT; i++)
            {
                arrange[i].Register();
                addCat[i].Register();
                close[i].Register();
                kill[i].Register();
                show[i].Register();
                hide[i].Register();
            }
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Unregister all the hotkeys
            autoDockKey.Unregister();
            restoreKey.Unregister();

            moveUpKey.Unregister();
            moveDownKey.Unregister();
            moveLeftKey.Unregister();
            moveRightKey.Unregister();
            sizeUpKey.Unregister();
            sizeDownKey.Unregister();

            arrangeKey.Unregister();
            closeKey.Unregister();
            killKey.Unregister();
            swapMainKey.Unregister();
            showKey.Unregister();
            hideKey.Unregister();

            for (int i = 0; i < CATEGORY_LIMIT; i++)
            {
                arrange[i].Unregister();
                addCat[i].Unregister();
                close[i].Unregister();
                kill[i].Unregister();
                show[i].Unregister();
                hide[i].Unregister();
            }

            if (autoDock)
            {
```

```
                dockUp.Unregister();
                dockDown.Unregister();
                dockLeft.Unregister();
                dockRight.Unregister();
                dockAltUp.Unregister();
                dockAltDown.Unregister();
                dockAltLeft.Unregister();
                dockAltRight.Unregister();
            }

        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            if (this.WindowState == FormWindowState.Minimized)
            {
                this.Hide();
                notifyIcon.Visible = true;
                notifyIcon.ShowBalloonTip(2000);
            }
        }

        private void Restore()
        {
            this.Show();
            this.WindowState = FormWindowState.Maximized;
            notifyIcon.Visible = false;
            RefreshTable(catIndex);
        }

        private void notifyIcon_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            Restore();
        }
        private void restoreToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Restore();
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void btnCloseAll_Click(object sender, EventArgs e)
        {
            DialogResult result = MessageBox.Show("Are you sure you want to close ALL
windows?", "Confirmation", MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
            if(result == DialogResult.OK)
            {
                this.WindowState = FormWindowState.Minimized;
                CloseAllWindows();
            }
        }

        private void btnKillAll_Click(object sender, EventArgs e)
```

```
        {
            DialogResult result = MessageBox.Show("Are you sure you want to kill ALL
processes? Unsaved progress will be lost.", "Confirmation", MessageBoxButtons.OKCancel,
MessageBoxIcon.Exclamation);
            if (result == DialogResult.OK)
            {
                this.WindowState = FormWindowState.Minimized;
                KillAllWindows();
            }
        }


        private bool IsAltTabWindow(IntPtr hwnd)
        {
            // Credits to Paul Dinham from Stack Overflow forums for providing C++ source.
            // Code was adapted to work in this particular C# application.

            // Dinham, P. (2011). c++ - Why does EnumWindows return more windows than I
expected?, Stack Overflow.
            // Available at: http://stackoverflow.com/questions/7277366/why-does-
enumwindows-return-more-windows-than-i-expected
            // [Accessed: 18 Dec 2013].

            TITLEBARINFO ti = new TITLEBARINFO();
            IntPtr hwndTry, hwndWalk = IntPtr.Zero;

            // If the window is hidden, it is not a real window.
            if (!IsWindowVisible(hwnd))
                return false;

            // If it's this application window, then just ignore it altogether.
            if (hwnd == this.Handle)
                return false;

            hwndTry = GetAncestor(hwnd, GA_ROOTOWNER);
            while (hwndTry != hwndWalk)
            {
                hwndWalk = hwndTry;
                hwndTry = GetLastActivePopup(hwndWalk);
                if (IsWindowVisible(hwndTry))
                    break;
            }
            if (hwndWalk != hwnd)
                return false;

            // the following removes some task tray programs and "Program Manager"
            ti.cbSize = (uint)Marshal.SizeOf(ti);
            GetTitleBarInfo(hwnd, ref ti);
            if ((ti.rgstate[0] & STATE_SYSTEM_INVISIBLE) != 0)
                return false;

            // Tool windows should not be displayed either, these do not appear in the
            // task bar.
            if ((GetWindowLong(hwnd, GWL_EXSTYLE) & WS_EX_TOOLWINDOW) != 0)
                return false;
```

```
        return true;
    }



    private void AutoArrange()
    {
        currentcIndex = 0;
        EnumWindows(new EnumWindowsProc(RearrangeWindow), IntPtr.Zero);
    }

    private void CloseAllWindows()
    {
        this.WindowState = FormWindowState.Minimized;
        EnumWindows(new EnumWindowsProc(CloseAll), IntPtr.Zero);
        RefreshTable(catIndex);
    }

    private void KillAllWindows()
    {
        EnumWindows(new EnumWindowsProc(KillAll), IntPtr.Zero);
        RefreshTable(catIndex);
    }

    private void ShowAllWindows()
    {
        this.WindowState = FormWindowState.Minimized;
        EnumWindows(new EnumWindowsProc(ShowAll), IntPtr.Zero);
    }

    private void HideAllWindows()
    {
        EnumWindows(new EnumWindowsProc(HideAll), IntPtr.Zero);
    }

    private bool ZOrderAdd(IntPtr hWnd, IntPtr lParam)
    {
        if (IsAltTabWindow(hWnd))
        {
            // Add windows to temporary group for determining z-order in auto-dock.
            tempGroup.AddWindow(CreateWindow(hWnd));
        }
        return true;
    }

    private bool CloseAll(IntPtr hWnd, IntPtr lParam)
    {
        Window temp = CreateWindow(hWnd);
        if (IsAltTabWindow(hWnd) || hiddenGroup.hasWindow(temp))
        {
            SendMessage(hWnd, WM_CLOSE, IntPtr.Zero, IntPtr.Zero);
            hiddenGroup.RemoveWindow(temp);
        }

        return true;
    }
```

```
private bool KillAll(IntPtr hWnd, IntPtr lParam)
{
    Window temp = CreateWindow(hWnd);
    if (IsAltTabWindow(hWnd) || hiddenGroup.hasWindow(temp))
    {
        int processID;
        GetWindowThreadProcessId(hWnd, out processID);
        hiddenGroup.RemoveWindow(temp);
        Process.GetProcessById(processID).Kill();
    }
    return true;
}

private void CloseCat(int index)
{
    if (windowGroups[index] != null)
    {
        this.WindowState = FormWindowState.Minimized;
        for (int i = 0; i < windowGroups[index].numberofWindows(); i++)
        {
            Window temp = CreateWindow(windowGroups[index].GetHandleAt(i));
            SendMessage(temp.Handle, WM_CLOSE, IntPtr.Zero, IntPtr.Zero);
            hiddenGroup.RemoveWindow(temp);
        }
    }
    RefreshTable(catIndex);
}

private void KillCat(int index)
{
    if (windowGroups[index] != null)
    {
        for (int i = 0; i < windowGroups[index].numberofWindows(); i++)
        {
            Window temp = CreateWindow(windowGroups[index].GetHandleAt(i));
            int processID;
            GetWindowThreadProcessId(temp.Handle, out processID);
            hiddenGroup.RemoveWindow(temp);
            Process.GetProcessById(processID).Kill();
        }
    }
    RefreshTable(catIndex);
}

private bool ShowAll(IntPtr hWnd, IntPtr lParam)
{
    Window temp = CreateWindow(hWnd);
    if (IsAltTabWindow(hWnd) || hiddenGroup.hasWindow(temp)) {
        hiddenGroup.RemoveWindow(temp);
        ShowWindow(hWnd, SW_RESTORE);
        ShowWindow(hWnd, SW_SHOW);
    }
    return true;
}
```

```
        private void ShowCat(int index)
        {
            if (windowGroups[index] != null)
            {
                this.WindowState = FormWindowState.Minimized;
                for (int i = 0; i < windowGroups[index].numberofWindows(); i++)
                {
                    Window temp = CreateWindow(windowGroups[index].GetHandleAt(i));
                    hiddenGroup.RemoveWindow(temp);
                    ShowWindow(temp.Handle, SW_RESTORE);
                    ShowWindow(temp.Handle, SW_SHOW);
                }
            }
        }

        private bool HideAll(IntPtr hWnd, IntPtr lParam)
        {
            if (IsAltTabWindow(hWnd))
            {
                hiddenGroup.AddWindow(CreateWindow(hWnd));
                ShowWindow(hWnd, SW_HIDE);
            }
            return true;
        }

        private void HideCat(int index)
        {
            if (windowGroups[index] != null)
            {
                for (int i = 0; i < windowGroups[index].numberofWindows(); i++)
                {
                    Window temp = CreateWindow(windowGroups[index].GetHandleAt(i));
                    hiddenGroup.AddWindow(temp);
                    ShowWindow(temp.Handle, SW_HIDE);
                }
            }
        }


        private Window CreateWindow(IntPtr hWnd)
        {
            int size = GetWindowTextLength(hWnd) + 1;
            StringBuilder sb = new StringBuilder(size);
            GetWindowText(hWnd, sb, size);
            RECT windowRect = new RECT();
            GetWindowRect(hWnd, ref windowRect);

            Window window = new Window(hWnd, sb.ToString(), windowRect.X, windowRect.Y,
windowRect.Width, windowRect.Height);
            return window;
        }


        private bool RearrangeWindow(IntPtr hWnd, IntPtr lParam)
        {
            // THIS FUNCTION IS FOR ARRANGING WINDOWS AND ADDING CATEGORIES
```

```
            if (IsAltTabWindow(hWnd))
            {
                Window currentWindow = CreateWindow(hWnd);
                bool alreadyExists = false;

                // Check if window already exists in any category.
                for (int i = 0; i < windowGroups.Length; i++ )
                {
                    if (windowGroups[i] == null)
                    {
                        continue;
                    }
                    else
                    {
                        if (windowGroups[i].hasWindow(currentWindow))
                        {
                            alreadyExists = true;
                        }
                    }
                }

                if (!alreadyExists)
                {
                    // CHECK IF FULL
                    while (windowGroups[currentcIndex] != null)
                    {
                        if (windowGroups[currentcIndex].isFull())
                        {
                            currentcIndex++;
                        }
                        else
                        {
                            break;
                        }
                    }

                    // If the current category is NULL, make this window the main window
in the new category.
                    if (windowGroups[currentcIndex] == null)
                    {
                        NewCategory(currentcIndex, currentWindow);
                    }
                    else
                    {
                        if (windowGroups[currentcIndex].MainWindow == null ||
windowGroups[currentcIndex].isEmpty())
                        {
                            // No main window! Assign this window as main window.
                            windowGroups[currentcIndex].MainWindow = currentWindow;
                        }
                    }

                    // Add current window to current category.
                    windowGroups[currentcIndex].AddWindow(currentWindow);
                    RearrangeCategory(currentcIndex);
```

```
                    // If current category is full, move on to the next.
                    if (windowGroups[currentcIndex].isFull())
                    {
                        currentcIndex++;
                    }
                }
            }

            return true;
        }

        private void RearrangeCategory(int index)
        {
            if (windowGroups[index] == null) { return; }
            UpdateCategories();
            int x = screenWidth / 2;
            int y = 0;
            int number = windowGroups[index].numberofWindows();

            switch (number)
            {
                case 1:
                    // set to cover the whole screen
                    SetWindowPos(windowGroups[index].GetHandleAt(0), IntPtr.Zero, 0, 0,
screenWidth, screenHeight, SWP_NOZORDER);
                    break;
                case 2:
                    for (int i = 0; i < number; i++)
                    {
                        // dock to both halves of screen
                        IntPtr currentHandle = windowGroups[index].GetHandleAt(i);
                        if (currentHandle == windowGroups[index].MainWindow.Handle)
                        {
                            SetWindowPos(currentHandle, IntPtr.Zero, 0, 0, x,
screenHeight, SWP_NOZORDER);
                        }
                        else
                        {
                            SetWindowPos(currentHandle, IntPtr.Zero, x, y, x,
screenHeight, SWP_NOZORDER);
                        }
                    }
                    break;
                case 3:
                    for (int i = 0; i < number; i++ )
                    {
                        // Dock 1 window left full height, 2 windows right with equally
divided heights
                        IntPtr currentHandle = windowGroups[index].GetHandleAt(i);
                        if (currentHandle == windowGroups[index].MainWindow.Handle)
                        {
                            SetWindowPos(currentHandle, IntPtr.Zero, 0, 0, screenWidth /
2, screenHeight, SWP_NOZORDER);
                        }
                        else
                        {
```

```
                            SetWindowPos(currentHandle, IntPtr.Zero, x, y, x,
screenHeight / 2, SWP_NOZORDER);
                            y += screenHeight / 2;
                        }
                    }
                    break;
                case 4:
                    // ==MAIN + 23:77 LAYOUT START== //
                    for (int i = 0; i < number; i++)
                    {
                        IntPtr currentHandle = windowGroups[index].GetHandleAt(i);
                        if (currentHandle == windowGroups[index].MainWindow.Handle)
                        {
                            SetWindowPos(currentHandle, IntPtr.Zero, 0, 0, screenWidth /
2, screenHeight, SWP_NOZORDER);
                        }
                        else if (y * 2 > screenHeight - y)
                        {
                            SetWindowPos(currentHandle, IntPtr.Zero, x, y, x,
screenHeight - y, SWP_NOZORDER);
                        }
                        else
                        {
                            // Plenty of space, continue to place windows in 23:77 ratio.
                            double tempY = (((double)23 / (double)77) * x);
                            int newY = (int)Math.Ceiling(tempY);
                            SetWindowPos(currentHandle, IntPtr.Zero, x, y, x, newY,
SWP_NOZORDER);
                            y = y + newY;
                        }
                    }
                    // ==MAIN + 23:77 LAYOUT END== //


                    /*
                    // ==GOLDEN SECTION LAYOUT START== //
                    // FOR TESTING AND CONTROL PURPOSES ONLY, DOES NOT FULLY WORK WITH
OTHER FUNCTIONS //
                    double goldenRatio = ((double)1 + (double)Math.Sqrt(5)) / 2;
                    // Height of screen = 1
                    // Width of screen = Golden Ratio
                    int a = (int)Math.Ceiling((double)screenWidth * (goldenRatio - 1));
                    int b = screenWidth - a;
                    int c = (int)Math.Ceiling((double)screenHeight * (goldenRatio - 1));
                    int d = screenHeight - c;
                    int e = (int)Math.Ceiling((double)b * (goldenRatio - 1));
                    for (int i = 0; i < number; i++)
                    {
                        IntPtr currentHandle = windowGroups[index].GetHandleAt(i);
                        switch (i)
                        {
                            case 0:
                                // 1st window
                                SetWindowPos(currentHandle, IntPtr.Zero, 0, 0, a,
screenHeight, SWP_NOZORDER);
                                break;
```

```
                            case 1:
                                // 2nd window
                                SetWindowPos(currentHandle, IntPtr.Zero, a, 0, b, c,
SWP_NOZORDER);
                                break;
                            case 2:
                                // 3rd window
                                SetWindowPos(currentHandle, IntPtr.Zero, a + b - e, c, e,
d, SWP_NOZORDER);
                                break;
                            case 3:
                                // 4th window
                                SetWindowPos(currentHandle, IntPtr.Zero, a, c, b - e, d,
SWP_NOZORDER);
                                break;
                        }
                    }
                    // ==GOLDEN SECTION LAYOUT END== //
                    */

                    break;
                default:
                    break;
            }

        }

        private void MoveActiveWindow(int direction)
        {
            // DIRECTION
            // 0 - Bottom
            // 1 - Right
            // 2 - Top
            // 3 - Left

            Window activeWindow = CreateWindow(GetForegroundWindow());
            if (IsAltTabWindow(activeWindow.Handle))
            {
                int movement = 50;
                switch (direction)
                {
                    case 0:
                        SetWindowPos(activeWindow.Handle, IntPtr.Zero, activeWindow.xPos,
activeWindow.yPos + movement, 0, 0, SWP_NOSIZE);
                        break;
                    case 1:
                        SetWindowPos(activeWindow.Handle, IntPtr.Zero, activeWindow.xPos
+ movement, activeWindow.yPos, 0, 0, SWP_NOSIZE);
                        break;
                    case 2:
                        SetWindowPos(activeWindow.Handle, IntPtr.Zero, activeWindow.xPos,
activeWindow.yPos - movement, 0, 0, SWP_NOSIZE);
                        break;
                    case 3:
                        SetWindowPos(activeWindow.Handle, IntPtr.Zero, activeWindow.xPos
- movement, activeWindow.yPos, 0, 0, SWP_NOSIZE);
```

```
                        break;
                    default:
                        // Do nothing.
                        break;
                }
            }


        }

        private void ResizeActiveWindow(int x)
        {
            Window activeWindow = CreateWindow(GetForegroundWindow());
            if (IsAltTabWindow(activeWindow.Handle))
            {
                int movement = 50;
                switch (x)
                {
                    case 0:
                        SetWindowPos(activeWindow.Handle, IntPtr.Zero, activeWindow.xPos,
activeWindow.yPos, activeWindow.Width + movement, activeWindow.Height + movement,
SWP_NOZORDER);
                        break;
                    case 1:
                        SetWindowPos(activeWindow.Handle, IntPtr.Zero, activeWindow.xPos,
activeWindow.yPos, activeWindow.Width - movement, activeWindow.Height - movement,
SWP_NOZORDER);
                        break;
                    default:
                        // Do nothing.
                        break;
                }
            }
        }

        private void AddtoGroup(Window window, int categoryIndex)
        {
            if (IsAltTabWindow(window.Handle))
            {
                UpdateCategories();
                if (windowGroups[categoryIndex] == null)
                {
                    NewCategory(categoryIndex, window);
                }

                if (windowGroups[categoryIndex].isEmpty())
                {
                    windowGroups[categoryIndex].MainWindow = window;
                }

                if (windowGroups[categoryIndex].AddWindow(window))
                {
                    // Remove from previous category, if exists.
                    for (int i = 0; i < windowGroups.Length; i++)
                    {
                        if (windowGroups[i] == null || i == categoryIndex)
```

```
                        {
                            continue;
                        }
                        else
                        {
                            if (windowGroups[i].hasWindow(window))
                            {
                                windowGroups[i].RemoveWindow(window);
                            }
                        }
                    }

                    UpdateCategories();
                    notifyIcon.ShowBalloonTip(2000, "Added window to category!", "'" +
window.Title + "' has been added to " + windowGroups[categoryIndex].CategoryName,
ToolTipIcon.None);
                    return;
                }
                else
                {
                    MessageBox.Show(new Form() { TopMost = true }, "Category is full or
window already exists.", "Unable to Add Window", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
                    return;
                }

            }
            else
            {
                MessageBox.Show(new Form() { TopMost = true }, "Select a valid active
window to be added to the category!", "Error: No Window Selected", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private void NewCategory(int index, Window main)
        {
            windowGroups[index] = new Category("Category #" + (index + 1), main,
LAYOUT_LIMIT);
            windowGroups[index].MainWindow = main;

            // Create CATEGORY button
            Button catButton = new Button();
            catButton.Text = windowGroups[index].CategoryName;
            catButton.Name = "btnCategory" + index;
            catButton.Width = 105;
            catButton.Height = 45;
            catButton.Font = new Font("Century Gothic", 10.0f, FontStyle.Bold);
            catButton.UseVisualStyleBackColor = true;
            catButton.Click += btnCategory_Click;
            toolTips.SetToolTip(catButton, "List all windows in this category.");

            // Create panel for category controls
            TableLayoutPanel catControl = new TableLayoutPanel();
            catControl.Name = "catControl" + index;
            catControl.AutoSize = true;
```

```
            catControl.GrowStyle = TableLayoutPanelGrowStyle.AddRows;
            catControl.Width = 25;
            catControl.Height = 45;
            catControl.Margin = Padding.Empty;

            // Rename button for category controls
            Button renameButton = new Button();
            renameButton.Name = "btnRenameCat" + index;
            renameButton.Image = WindowsFormsApplication1.Properties.Resources.smalledit;
            renameButton.ImageAlign = ContentAlignment.MiddleCenter;
            renameButton.Width = 22;
            renameButton.Height = 22;
            renameButton.UseVisualStyleBackColor = true;
            renameButton.Margin = new Padding(0, 3, 0, 0);
            renameButton.Click += btnRenameCat_Click;
            toolTips.SetToolTip(renameButton, "Rename this category.");

            // Delete category button for category controls
            Button catDelButton = new Button();
            catDelButton.Name = "btnDeleteCat" + index;
            catDelButton.Image =
WindowsFormsApplication1.Properties.Resources.smalldelete;
            catDelButton.ImageAlign = ContentAlignment.MiddleCenter;
            catDelButton.Width = 22;
            catDelButton.Height = 22;
            catDelButton.UseVisualStyleBackColor = true;
            catDelButton.Margin = Padding.Empty;
            catDelButton.Click += btnDelCat_Click;
            toolTips.SetToolTip(catDelButton, "Delete this category.");

            // Add the controls in their respective places
            panelCategory.Controls.Add(catButton);
            panelCategory.Controls.SetChildIndex(catButton, (2 * index) + 3);
            catControl.Controls.Add(renameButton);
            catControl.Controls.Add(catDelButton);
            panelCategory.Controls.Add(catControl);
            panelCategory.Controls.SetChildIndex(catControl,
panelCategory.Controls.GetChildIndex(catButton) + 1);
        }

        private void UpdateCategories()
        {
            // Loop through all categories.
            for (int i = 0; i < windowGroups.Length; i++)
            {
                if (windowGroups[i] != null)
                {
                    int counter = 0;
                    int limit = windowGroups[i].numberofWindows();
                    while (counter < limit)
                    {
                        Window currentWindow =
CreateWindow(windowGroups[i].GetHandleAt(counter));
                        // Check if the window still exists.
                        if (!IsAltTabWindow(currentWindow.Handle)
&& !hiddenGroup.hasWindow(currentWindow))
```

```
                        {
                            windowGroups[i].RemoveAt(counter);
                            limit--;
                            // Everytime a window is removed, it automatically shifts the
objects up the list.
                            // Don't increment counter because the next object will be
shifted to the current index.
                        }
                        else
                        {
                            counter++;
                        }
                    }
                }
            }
        }

        private void RefreshTable(int showIndex)
        {
            previewGrid.Controls.Clear();
            previewGrid.ColumnStyles.Clear();
            activeButtons.Clear();

            // Unregister all thumbnails
            for (int i = 0; i < preview.Length; i++)
            {
                if (preview[i] != IntPtr.Zero) { DwmUnregisterThumbnail(preview[i]); }
            }


            // Just another layer of handling for NullException. Maybe use try/catch next
time?
            if (windowGroups[showIndex] != null)
            {
                UpdateCategories();
                lblCurrentCat.Text = windowGroups[showIndex].CategoryName;
                int number = windowGroups[showIndex].numberofWindows();

                // Prepare TableLayoutPanel
                for (int i = 0; i < number; i++)
                {
                    Window currentWindow =
CreateWindow(windowGroups[showIndex].GetHandleAt(i));
                    FlowLayoutPanel section = new FlowLayoutPanel() { Width =
previewGrid.Width / number, Height = (previewGrid.Height - 100) };
                    previewGrid.Controls.Add(section, i, 0);
                    FlowLayoutPanel controls = new FlowLayoutPanel() { Width =
previewGrid.Width / number, Height = 100 };


                    Button windowIcon = new Button() { Width = 48, Height = 48,
UseVisualStyleBackColor = true };
                    windowIcon.Name = "btnSwitchTo" + i;
                    try
                    {
                        windowIcon.Image = GetIcon(currentWindow.Handle).ToBitmap();
```

```
                    }
                    catch (Exception e)
                    {
                        // Null icon, simply leave it and don't assign.
                    }
                    windowIcon.ImageAlign = ContentAlignment.MiddleCenter;
                    windowIcon.Click += windowIcon_Click;
                    controls.Controls.Add(windowIcon);

                    Label tempLabel = new Label();
                    tempLabel.ForeColor = Color.Red;
                    int blah = (previewGrid.Width / number) - 200;
                    tempLabel.MaximumSize = new Size(blah, 0);
                    tempLabel.AutoSize = true;
                    tempLabel.Text = currentWindow.Title;
                    tempLabel.Margin = new Padding(0, 10, 0, 0);
                    controls.Controls.Add(tempLabel);

                    RadioButton makeActive = new RadioButton() { Width = 42, Height = 42,
UseVisualStyleBackColor = true };
                    makeActive.Appearance = Appearance.Button;
                    makeActive.Name = "btnMakeActive" + i;
                    makeActive.Image =
WindowsFormsApplication1.Properties.Resources.blackactive;
                    makeActive.ImageAlign = ContentAlignment.MiddleCenter;
                    if (windowGroups[catIndex].GetHandleAt(i) ==
windowGroups[catIndex].MainWindow.Handle) { makeActive.Checked = true; }
                    makeActive.Click += btnMakeActive_Click;
                    activeButtons.Add(makeActive);
                    controls.Controls.Add(makeActive);

                    Button closeThis = new Button() { Width = 42, Height = 42,
UseVisualStyleBackColor = true };
                    closeThis.Name = "btnCloseThis" + i;
                    closeThis.Image =
WindowsFormsApplication1.Properties.Resources.blackcross;
                    closeThis.ImageAlign = ContentAlignment.MiddleCenter;
                    closeThis.Click += btnCloseThis_Click;
                    controls.Controls.Add(closeThis);

                    Button killThis = new Button() { Width = 42, Height = 42,
UseVisualStyleBackColor = true };
                    killThis.Name = "btnKillThis" + i;
                    killThis.Image =
WindowsFormsApplication1.Properties.Resources.blackskull;
                    killThis.ImageAlign = ContentAlignment.MiddleCenter;
                    killThis.Click += btnKillThis_Click;
                    controls.Controls.Add(killThis);

                    previewGrid.Controls.Add(controls, i, 1);
                }

                for (int i = 0; i < number; i++)
                {
                    // Credits to Bart De Smet for a wonderful tutorial on DWM API and
window previews.
```

```
                    // Codes from some parts of the tutorial were taken and modified
                    // to suit this application's purposes.

                    // De Smet, B. (2006). Programming the Windows Vista DWM in C#.
[online] B# .NET Blog.
                    // Available at:
http://bartdesmet.net/blogs/bart/archive/2006/10/05/4495.aspx
                    // [Accessed 22 Feb 2014].

                    IntPtr thumb;
                    DwmRegisterThumbnail(this.Handle,
windowGroups[showIndex].GetHandleAt(i), out thumb);
                    preview[i] = thumb;

                    DWM_THUMBNAIL_PROPERTIES props = new DWM_THUMBNAIL_PROPERTIES();
                    props.dwFlags = DWM_TNP_VISIBLE | DWM_TNP_RECTDESTINATION |
DWM_TNP_OPACITY;

                    props.fVisible = true;
                    props.opacity = 255;

                    Control control = previewGrid.GetControlFromPosition(i, 0);
                    // Set position of the window snapshot, with an offset based on the
table's position.
                    props.rcDestination = new RECT(control.Left + previewGrid.Location.X,
control.Top + previewGrid.Location.Y, control.Right, control.Bottom);

                    DwmUpdateThumbnailProperties(preview[i], ref props);
                }


            }

        }


        protected override void WndProc(ref Message m)
        {
            // Handles all the hotkeys
            if (m.Msg == HotkeyConstants.WM_HOTKEY_MSG_ID)
            {
                Keys key = (Keys)(((int)m.LParam >> 16) & 0xFFFF);
                int modifier = (int)m.LParam & 0xFFFF;
                if (modifier == arrangeKey.modifier && key == arrangeKey.key)
                {
                    AutoArrange();
                }
                else if ((modifier == restoreKey.modifier) && key == restoreKey.key)
                {
                    if (this.WindowState == FormWindowState.Maximized)
                    {
                        this.WindowState = FormWindowState.Minimized;
                    }
                    else
                    {
                        Restore();
```

```
                }
            }
            else if ((modifier == closeKey.modifier) && key == closeKey.key)
            {
                DialogResult result = MessageBox.Show("Are you sure you want to close
ALL windows?", "Confirmation", MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
                if (result == DialogResult.OK)
                {
                    this.WindowState = FormWindowState.Minimized;
                    CloseAllWindows();
                }
            }
            else if ((modifier == killKey.modifier) && key == killKey.key)
            {
                DialogResult result = MessageBox.Show("Are you sure you want to kill
ALL processes? Unsaved progress will be lost.", "Confirmation",
MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
                if (result == DialogResult.OK)
                {
                    this.WindowState = FormWindowState.Minimized;
                    KillAllWindows();
                }
            }
            else if ((modifier == moveUpKey.modifier) && key == moveUpKey.key)
            {
                MoveActiveWindow(2);
            }
            else if ((modifier == moveRightKey.modifier) && key == moveRightKey.key)
            {
                MoveActiveWindow(1);
            }
            else if ((modifier == moveDownKey.modifier) && key == moveDownKey.key)
            {
                MoveActiveWindow(0);
            }
            else if ((modifier == moveLeftKey.modifier) && key == moveLeftKey.key)
            {
                MoveActiveWindow(3);
            }
            else if ((modifier == sizeUpKey.modifier) && key == sizeUpKey.key)
            {
                ResizeActiveWindow(0);
            }
            else if ((modifier == sizeDownKey.modifier) && key == sizeDownKey.key)
            {
                ResizeActiveWindow(1);
            }
            else if ((modifier == showKey.modifier) && key == showKey.key)
            {
                ShowAllWindows();
            }
            else if ((modifier == hideKey.modifier) && key == hideKey.key)
            {
                HideAllWindows();
            }
            else if ((modifier == swapMainKey.modifier) && key == swapMainKey.key)
```

```
                {
                    Window newMain = CreateWindow(GetForegroundWindow());
                    bool alreadyExists = false;
                    int index = 0;

                    // Check if window already exists in any category.
                    for (int i = 0; i < windowGroups.Length; i++)
                    {
                        if (windowGroups[i] == null)
                        {
                            continue;
                        }
                        else
                        {
                            if (windowGroups[i].hasWindow(newMain))
                            {
                                alreadyExists = true;
                                index = i;
                            }
                        }
                    }
                    if (alreadyExists)
                    {
                        windowGroups[index].MainWindow = newMain;
                        RearrangeCategory(catIndex);
                    }
                    else { MessageBox.Show(new Form() { TopMost = true }, "The selected
window does not exist in any category!"); }

                }
                else if ((modifier == autoDockKey.modifier) && key == autoDockKey.key)
                {
                    if (autoDock)
                    {
                        // Switch off auto-dock.
                        autoDock = false;
                        dockUp.Unregister();
                        dockDown.Unregister();
                        dockLeft.Unregister();
                        dockRight.Unregister();
                        dockAltUp.Unregister();
                        dockAltDown.Unregister();
                        dockAltLeft.Unregister();
                        dockAltRight.Unregister();
                        adWindows[0].Clear();
                        adWindows[1].Clear();
                        adWindows[2].Clear();
                        lblADStatus.Text = "Off";
                        lblADStatus.ForeColor = Color.Red;
                        notifyIcon.ShowBalloonTip(2000, "Autodock is OFF!", "To turn this
on again, press Ctrl + Shift + Tab.", ToolTipIcon.None);
                    }
                    else
                    {
                        // Switch on auto-dock.
                        autoDock = true;
```

```
                        adWindows[0] = new Category("Left", 8);
                        adWindows[1] = new Category("Middle", 8);
                        adWindows[2] = new Category("Right", 8);
                        dockUp.Register();
                        dockDown.Register();
                        dockLeft.Register();
                        dockRight.Register();
                        dockAltUp.Register();
                        dockAltDown.Register();
                        dockAltLeft.Register();
                        dockAltRight.Register();
                        lblADStatus.Text = "On";
                        lblADStatus.ForeColor = Color.Green;
                        notifyIcon.ShowBalloonTip(2000, "Autodock is ON!", "Arrow keys
now dock windows to the screen. To turn this off, press Ctrl + Shift + Tab.",
ToolTipIcon.None);
                    }
                }

                // Auto-dock processing.
                if (autoDock)
                {
                    if ((modifier == dockUp.modifier && key == dockUp.key) ||
                        (modifier == dockDown.modifier && key == dockDown.key) ||
                        (modifier == dockLeft.modifier && key == dockLeft.key) ||
                        (modifier == dockRight.modifier && key == dockRight.key) ||
                        (modifier == dockAltUp.modifier && key == dockAltUp.key) ||
                        (modifier == dockAltDown.modifier && key == dockAltDown.key) ||
                        (modifier == dockAltLeft.modifier && key == dockAltLeft.key) ||
                        (modifier == dockAltRight.modifier && key == dockAltRight.key))
                    {
                        int col = 1;
                        if (key == dockLeft.key)
                        {
                            col = 0;
                        }
                        else if (key == dockRight.key)
                        {
                            col = 2;
                        }

                        Window activeWindow = CreateWindow(GetForegroundWindow());
                        if (IsAltTabWindow(activeWindow.Handle))
                        {
                            for (int i = 0; i < 3; i++)
                            {
                                // If this window already exists in any column, remove it.
                                if (adWindows[i].hasWindow(activeWindow))
{ adWindows[i].RemoveWindow(activeWindow); }
                                // Remove non-existent windows from columns.
                                for (int j = 0; j < adWindows[i].numberofWindows(); j++)
                                {
                                    if (!IsAltTabWindow(adWindows[i].GetHandleAt(j)))
{ adWindows[i].RemoveWindow(adWindows[i].GetWindowAt(j)); }
                                }
                            }
```

```
                            // Add the window to the column.
                            if (key == dockUp.key)
{ adWindows[col].AddtoFirst(activeWindow); }
                            else { adWindows[col].AddWindow(activeWindow); }

                            int xmod = 2;
                            if (!adWindows[1].isEmpty()) { xmod = 3; }

                            // Update all columns.
                            for (int i = 0; i < 3; i++)
                            {
                                int number = adWindows[i].numberofWindows();
                                for (int j = 0; j < number; j++)
                                {
                                    int x = i;
                                    if (xmod == 2 && i == 2) { x = i - 1; }
                                    ShowWindow(adWindows[i].GetHandleAt(j), SW_RESTORE);
                                    SetWindowPos(adWindows[i].GetHandleAt(j),  IntPtr.Zero,
(screenWidth / xmod) * x, (screenHeight / number) * j, screenWidth / xmod, screenHeight /
number, SWP_NOZORDER);
                                }
                            }


                            if (modifier == dockAltUp.modifier || modifier ==
dockAltDown.modifier || modifier == dockAltLeft.modifier || modifier ==
dockAltRight.modifier)
                            {
                                // SWITCH TO NEXT WINDOW //
                                EnumWindows(new EnumWindowsProc(ZOrderAdd), IntPtr.Zero);
                                bool isLastWindow = false;
                                IntPtr storedHandle = activeWindow.Handle;
                                IntPtr tempHandle = storedHandle;
                                int counter = 0;

                                // Loop up the Z-Order.
                                while (!isLastWindow)
                                {
                                    IntPtr nextHandle = GetWindow(tempHandle,
GetWindow_Cmd.GW_HWNDPREV);

                                    // Is that a window in the group?
                                    if (tempGroup.hasWindow(CreateWindow(nextHandle)))
                                    {
                                        // Yes? Then switch to that window and continue,
add to counter.

                                        counter++;
                                        tempHandle = nextHandle;
                                    }
                                    else
                                    {
                                        // No? Then check if it is the top of the Z-Order.
                                        if (nextHandle == null || nextHandle ==
IntPtr.Zero)

                                        {
```

```
                                        // If it is the end, then check if it is
equal to the number of windows we have minus one.
                                        if (counter == (tempGroup.numberofWindows() -
1))
                                        {
                                            // If it is equal, it means all other
windows are on top of this window, so this is the lowest of the group.
                                            SetForegroundWindow(storedHandle);
                                            isLastWindow = true;
                                        }
                                        else
                                        {
                                            // If it is not equal, means there is
still a window below. Loop down the Z-Order until a valid window is found.
                                            IntPtr switcher = GetWindow(storedHandle,
GetWindow_Cmd.GW_HWNDNEXT);
                                            while
(!tempGroup.hasWindow(CreateWindow(switcher)))
                                            {
                                                switcher = GetWindow(switcher,
GetWindow_Cmd.GW_HWNDNEXT);
                                            }
                                            // Loop exits when found a valid window.
Now switch and start over.
                                            storedHandle = switcher;
                                            tempHandle = storedHandle;
                                            counter = 0;
                                        }

                                    }
                                    else
                                    {
                                        // Not the top of the Z-Order, switch and
continue, do not add to counter.
                                        tempHandle = nextHandle;
                                    }
                                }
                            }
                            // Clear the temporary group.
                            tempGroup.Clear();
                        }
                    }
                }
            }

            // Loop through category-based Hotkeys
            for (int i = 0; i < CATEGORY_LIMIT; i++)
            {
                if (modifier == addCat[i].modifier && key == addCat[i].key)
                {
                    Window activeWindow = CreateWindow(GetForegroundWindow());
                    AddtoGroup(activeWindow, i);
                    break;
                }
                if (modifier == arrange[i].modifier && key == arrange[i].key)
                {
```

```
                        RearrangeCategory(i);
                        break;
                    }
                    if (modifier == show[i].modifier && key == show[i].key)
                    {
                        ShowCat(i);
                        break;
                    }
                    if (modifier == hide[i].modifier && key == hide[i].key)
                    {
                        HideCat(i);
                        break;
                    }
                    if (modifier == close[i].modifier && key == close[i].key)
                    {
                        if (windowGroups[i] == null) { return; }
                        if (windowGroups[i].isEmpty()) { return; }
                        DialogResult result = MessageBox.Show("Are you sure you want to
close windows under '" + windowGroups[i].CategoryName + "'?", "Confirmation",
MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
                        if (result == DialogResult.OK)
                        {
                            CloseCat(i);
                        }
                        break;
                    }
                    if (modifier == kill[i].modifier && key == kill[i].key)
                    {
                        if (windowGroups[i] == null) { return; }
                        if (windowGroups[i].isEmpty()) { return; }
                        DialogResult result = MessageBox.Show("Are you sure you want to
kill processes under '" + windowGroups[i].CategoryName + "'?", "Confirmation",
MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
                        if (result == DialogResult.OK)
                        {
                            KillCat(i);
                        }
                        break;
                    }
                }
            }
            base.WndProc(ref m);
        }


        private void btnClose_Click(object sender, EventArgs e)
        {
            this.WindowState = FormWindowState.Minimized;
        }


        private void btnCategory_Click(object sender, EventArgs e)
        {
            Button someButton = sender as Button;
            String s = someButton.Name;
            int groupIndex = (int)Char.GetNumericValue(s[11]);
```

```
            catIndex = groupIndex;
            RefreshTable(groupIndex);
        }

        private void btnDelCat_Click(object sender, EventArgs e)
        {
            Button someButton = sender as Button;
            String s = someButton.Name;
            int groupIndex = (int)Char.GetNumericValue(s[12]);

            windowGroups[groupIndex] = null;

            foreach (Control item in panelCategory.Controls.OfType<TableLayoutPanel>())
            {
                if (item.Name == ("catControl" + groupIndex))
                    panelCategory.Controls.Remove(item);
            }

            foreach (Control item in panelCategory.Controls.OfType<Button>())
            {
                if (item.Name == ("btnCategory" + groupIndex))
                    panelCategory.Controls.Remove(item);
            }

        }

        private void btnRenameCat_Click(object sender, EventArgs e)
        {
            Button someButton = sender as Button;
            String s = someButton.Name;
            int groupIndex = (int)Char.GetNumericValue(s[12]);

            foreach (Control item in panelCategory.Controls.OfType<Button>())
            {
                if (item.Name == ("btnCategory" + groupIndex))
                {
                    TextBox renamer = new TextBox();
                    renamer.Name = "txtRenamer" + groupIndex;
                    renamer.Width = 130;
                    renamer.Text = item.Text;
                    renamer.MaxLength = 20;
                    renamer.KeyUp += Rename_KeyUp;
                    renamer.LostFocus += Rename_LostFocus;
                    panelCategory.Controls.Add(renamer);
                    panelCategory.Controls.SetChildIndex(renamer,
panelCategory.Controls.GetChildIndex(item) + 2);
                    toolTips.SetToolTip(renamer, "Enter the name for this category.");
                    renamer.Focus();
                }

            }
        }

        private void Rename_KeyUp(object sender, KeyEventArgs e)
        {
            TextBox renamer = sender as TextBox;
```

```
            String s = renamer.Name;
            int groupIndex = (int)Char.GetNumericValue(s[10]);
            if (e.KeyCode == Keys.Enter)
            {
                foreach (Control item in panelCategory.Controls.OfType<Button>())
                {
                    if (item.Name == ("btnCategory" + groupIndex))
                    {
                        item.Text = renamer.Text;
                        windowGroups[groupIndex].CategoryName = renamer.Text;
                    }
                }
                if (catIndex == groupIndex) { lblCurrentCat.Text =
windowGroups[catIndex].CategoryName; }
                panelCategory.Controls.Remove(renamer);
                e.Handled = true;
            }
        }

        private void Rename_LostFocus(object sender, EventArgs e)
        {
            TextBox renamer = sender as TextBox;
            panelCategory.Controls.Remove(renamer);
        }

        private void btnAutoArrange_Click(object sender, EventArgs e)
        {
            AutoArrange();
        }

        private void btnArrangeCat_Click(object sender, EventArgs e)
        {
            RearrangeCategory(catIndex);
        }

        private void btnCloseCat_Click(object sender, EventArgs e)
        {
            if (windowGroups[catIndex] == null) { return; }
            if (windowGroups[catIndex].isEmpty()) { return; }
            DialogResult result = MessageBox.Show("Are you sure you want to close windows
under '" + windowGroups[catIndex].CategoryName + "'?", "Confirmation",
MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
            if (result == DialogResult.OK)
            {
                CloseCat(catIndex);
            }

        }

        private void btnCloseThis_Click(object sender, EventArgs e)
        {
            Button button = sender as Button;
            String s = button.Name;
            int groupIndex = (int)Char.GetNumericValue(s[12]);
            Window closingWindow =
CreateWindow(windowGroups[catIndex].GetHandleAt(groupIndex));
```

```
            DialogResult result = MessageBox.Show("Are you sure you want to close '" +
closingWindow.Title + "'?", "Confirmation", MessageBoxButtons.OKCancel,
MessageBoxIcon.Exclamation);
            if (result == DialogResult.OK)
            {
                SendMessage(closingWindow.Handle, WM_CLOSE, IntPtr.Zero, IntPtr.Zero);
                RefreshTable(catIndex);
            }

        }

        private void btnKillThis_Click(object sender, EventArgs e)
        {
            Button button = sender as Button;
            String s = button.Name;
            int groupIndex = (int)Char.GetNumericValue(s[11]);
            Window killingWindow =
CreateWindow(windowGroups[catIndex].GetHandleAt(groupIndex));
            DialogResult result = MessageBox.Show("Are you sure you want to terminate '"
+ killingWindow.Title + "'? Unsaved progress will be lost.", "Confirmation",
MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
            if (result == DialogResult.OK)
            {
                int processID;
                GetWindowThreadProcessId(killingWindow.Handle, out processID);
                Process.GetProcessById(processID).Kill();
                RefreshTable(catIndex);
            }
        }

        private void btnKillCat_Click(object sender, EventArgs e)
        {
            if (windowGroups[catIndex] == null) { return; }
            if (windowGroups[catIndex].isEmpty()) { return; }
            DialogResult result = MessageBox.Show("Are you sure you want to kill
processes under '" + windowGroups[catIndex].CategoryName + "'?", "Confirmation",
MessageBoxButtons.OKCancel, MessageBoxIcon.Exclamation);
            if (result == DialogResult.OK)
            {
                KillCat(catIndex);
            }
        }

        private void btnMakeActive_Click(object sender, EventArgs e)
        {
            RadioButton button = sender as RadioButton;
            String s = button.Name;
            int groupIndex = (int)Char.GetNumericValue(s[13]);

            if (button.Checked)
            {
                foreach (RadioButton r in activeButtons)
                {
                    if (r == button)
                    {
                        continue;
```

```
                    }
                    r.Checked = false;
                }

                Window newMain =
CreateWindow(windowGroups[catIndex].GetHandleAt(groupIndex));
                windowGroups[catIndex].MainWindow = newMain;
                RearrangeCategory(catIndex);
            }
        }

        private void windowIcon_Click(object sender, EventArgs e)
        {
            Button someButton = sender as Button;
            String s = someButton.Name;
            int groupIndex = (int)Char.GetNumericValue(s[11]);
            this.WindowState = FormWindowState.Minimized;
            IntPtr targetHandle = windowGroups[catIndex].GetHandleAt(groupIndex);
            ShowWindow(targetHandle, SW_RESTORE);
            SetForegroundWindow(targetHandle);
            ShowWindow(targetHandle, SW_MAXIMIZE);
        }


        public Icon GetIcon(IntPtr hwnd)
        {
            // Credits to Jani Hartikainen for ready-made source code on obtaining window
icons.
            // It covers every possible method to obtain every possible icon. It is the
best solution,
            // I don't think I can even adapt or improve it.

            // Hartikainen, J. (2007). Find an application's icon with WinAPI. [online]
CodeUtopia.
            // Available at: http://codeutopia.net/blog/2007/12/18/find-an-applications-
icon-with-winapi/
            // [Accessed 22 Feb 2014].

            IntPtr iconHandle = SendMessage(hwnd, WM_GETICON, ICON_SMALL2, 0);
            if (iconHandle == IntPtr.Zero)
                iconHandle = SendMessage(hwnd, WM_GETICON, ICON_SMALL, 0);
            if (iconHandle == IntPtr.Zero)
                iconHandle = SendMessage(hwnd, WM_GETICON, ICON_BIG, 0);
            if (iconHandle == IntPtr.Zero)
                iconHandle = GetClassLongPtr(hwnd, GCL_HICON);
            if (iconHandle == IntPtr.Zero)
                iconHandle = GetClassLongPtr(hwnd, GCL_HICONSM);

            if (iconHandle == IntPtr.Zero)
                return null;

            Icon icn = Icon.FromHandle(iconHandle);

            return icn;
        }
```

```
        private void btnSettings_Click(object sender, EventArgs e)
        {
            settingsForm.Hide();
            settingsForm.Show();
        }

        private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            settingsForm.Hide();
            settingsForm.Show();
        }

        private void btnVisible_Click(object sender, EventArgs e)
        {
            if (windowGroups[catIndex] == null) { return; }
            if (windowGroups[catIndex].isEmpty()) { return; }
            ShowCat(catIndex);
        }

        private void btnHidden_Click(object sender, EventArgs e)
        {
            if (windowGroups[catIndex] == null) { return; }
            if (windowGroups[catIndex].isEmpty()) { return; }
            HideCat(catIndex);
        }
    }
}
```

Form2.cs (Settings) Class

```
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form2 : Form
    {
        // The path to the registry key where Windows looks for startup applications
(current user only)
        RegistryKey rk =
Registry.CurrentUser.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
true);
```

```
        public Form2()
        {
            InitializeComponent();
            // Check if already running in startup
            if (rk.GetValue("Window Management Tool") != null)
            {
                // If exists, tick the checkbox.
                chkStartup.Checked = true;
            }
        }

        private void btnOK_Click(object sender, EventArgs e)
        {
            if (chkStartup.Checked)
            {
                // Add the value in the registry
                rk.SetValue("Window Management Tool",
Application.ExecutablePath.ToString());
            }
            else
            {
                // Remove the value from the registry
                rk.DeleteValue("Window Management Tool", false);
            }
            this.Hide();
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.Hide();
        }

        private void Form2_FormClosing(object sender, FormClosingEventArgs e)
        {
            this.Hide();
            e.Cancel = true;
        }

    }
}
```