COSC460 Honours Report

Improving Task Switching Interfaces

Keith Humm

khu23@student.canterbury.ac.nz

Department of Computer Science and Software Engineering

University of Canterbury, Christchurch, New Zealand

November 8, 2007

Supervisor: Assoc. Prof. **Dr. Andy Cockburn**

andy@cosc.canterbury.ac.nz

Abstract

Most interfaces for switching between tasks require slow, visual searches of candidates, where each candidate must be scanned in turn. We developed a logging tool for task switching actions and undertook a log-based study with eleven participants to empirically characterise task-switching behaviour, finding Zipfian distributions for window and application switching, and significant differences in interface use between single and dual monitor users. We then used this data to assist in designing a new interface (Spatially COnsistent Thumbnail Zones or SCOTZ) to allow rapid task-switching performance by utilising spatial memory. In a formal evaluation against three mainstream interfaces under four workspace conditions, SCOTZ attained the lowest mean times for all conditions and scaled better with workspace load, obtaining a significant difference under high workspace load. We conclude with a recommendation for using SCOTZ at all workspace loads, and suggest it be implemented and tested in a real-world environment.

Acknowledgments

The author would like to thank Andy Cockburn for his guidance, support and supervision throughout this year. He also thanks all participants involved in the numerous experiments over the course of this research for their time and feedback. Thanks must also go to his family, friends, staff members and fellow students who have shown interest, given support, and participated in many a (heated) discussion. Finally, to the honours class of 2007: good luck with all your future endeavours and thank you all for a great year.

Contents

List of Figures

Introduction 1

This project attempts to gain an empirical understanding of task switching from which to base future design decisions on. We describe research conducted by others relating to task switching in Chapter 2, noting especially the large number of high-level *task*-oriented switching interfaces. Chapter 3 reports on the deployment and results of a log-based study empirically characterising task switching interfaces with good results. While we did not gather all the data we had intended to due to large amounts of noise, the data we did manage to gather proved an excellent resource to base design decisions on. We used these design decisions while implementing Spatially COnsistent Thumbnails (SCOTZ), an interface to support task switching through spatial memory, and minimise visual search time in the task switching process. We then report on the findings of an empirical study pitting SCOTZ against three mainstream task switching interfaces where we gained favourable results. Finally, we conclude by giving some avenues for future research for spatially stable task switching interfaces based on the SCOTZ principle. The following paragraphs give a brief history into the problem of task switching and where it began.

The 'window' desktop paradigm is shared by many different graphical user interfaces, most of which share a common set of methods by which to switch windows, applications and tasks. Many elements to this common set were developed for the Xerox Star and have remained largely unchanged to date — minimise/maximise buttons and desktop icons for example (Bewley *et al.* 1983, Harslem & Nelson 1982). While the original version of the Star managed multiple application windows by simply tiling them to fit the leftover screen area (Lineback 2006), the Apple Macintosh (circa 1984) allowed windows to overlap and thus gave birth to the problem of switching between these overlapping windows. This was a non-issue with the original Macintosh, as only a singe true application could be used at one time; however, the advent of multi-tasking desktop operating environments such as Microsoft Windows and Macintosh MultiFinder necessitated the widespread introduction of window and application switching interfaces.

Looking at multi-tasking from the user's perspective, we use desktop computers today for many different purposes, such as sending photos to friends by e-mail, building and deploying websites, and compiling business reports. Each of these activities can be defined as a high-level *task*: a global and very abstract concept that involves both a goal to accomplish, and a purpose for doing so. Each task also involves a means (or a number of means) by which to accomplish the task that may involve multiple different steps (tasks within tasks) — the different steps potentially involving different software or hardware available to the user, for example “drawing the sales figures chart in excel” or “writing the executive summary”. This definition of a task fits well with the *GOMS* model explained in Chapter 2.

Unfortunately, while we often maintain high level definitions of tasks in our own minds, computer systems seldom support them. Interruptions often occur during workflows, as explained further in Chapter 2, and task switching interfaces current are burdened by having to support these interruptions whilst attempting to provide high level grouping. This results in wasted space for additional widgets, unnecessary clicks or keystrokes, and cluttered interfaces requiring slow visual searches to find selection targets. A modern 17-inch liquid-crystal display (LCD) with a resolution of approximately 100 dots per inch (dpi) struggles to display two pages of a book with legibility while a small book can easily outshine this. Yet most task switching interfaces do not even attempt to utilise more than a small portion of this very limited display space. We set out to design SCOTZ to solve this problem and present a better task switching interface that becomes second nature and does not require visual searches.

Related Work 2

This chapter outlines previous work relating to task switching and task switching interfaces. First we review research investigating what tasks are, how users define them, and how workflow can be interrupted. We then describe log-based empirical methods for collecting task switching data and their findings, followed by an overview of current mainstream windowing environments together with research systems and their evaluations. Finally, we briefly investigate the role of spatial memory in navigable interfaces.

2.1 Tasks and Workflows

2.1.1 Defining Tasks

Based on applied information psychology, the *GOMS* model by Card *et al.* (1983) defines a user's cognitive structure for an activity as consisting of four elementary components: *a*) a set of Goals, *b*) a set of Operators, *c*) a set of Methods for achieving the goals, and *d*) a set of Selection rules for choosing among competing methods for goals. These four components form the basis for all tasks and sub-tasks. For example, the goal `'Edit Manuscript'` consists of sub-tasks such as `'Get Next Page'` and `'Add a Quaver Rest Here'`, which further break down into *sub* sub-tasks called *unit tasks* (e.g. `'Press Quaver Rest Button'`) — the exact unit tasks being decided upon by the user. Indeed, unit tasks are often part of *free-form* tasks—arbitrarily assigned *on-the-fly* by the user—that Iqbal & Bailey (2007) claim are the most common computing tasks. This model fits well with the simplified model presented in Chapter 1, where the *means* is formed by *operators* and *methods* in *GOMS* terminology.

Bardram *et al.* (2006) developed a model they call "Activity-Based Computing" (ABC). While their study primarily focuses on mobility around office systems, ABC stands on its own as another complete definition for tasks. In ABC, tasks (*activities*) are compositions of *services* and *data* related by work. *Services* map to applications and *data* are the documents, files or windows in use. Each service uses a different set of data, but two may share the same data — in Figure 2.1 the service *PDF Viewer* is associated with three pieces of data. *Activities* in ABC loosely correspond to high-level *goals* in *GOMS*, while *services* correspond to the *selected methods* of a *GOMS* sub-task at the application level. The ABC model fits more closely with current window

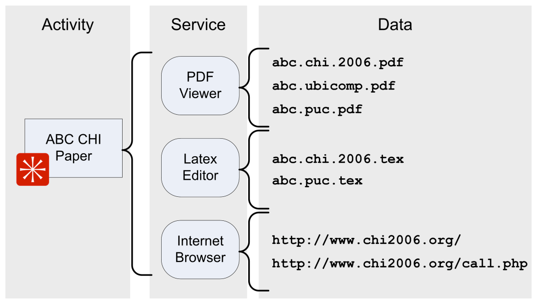


Figure 2.1: Activity-Based Computing Task, "ACM CHI Paper", composed of three services each with separate data. Figure taken from Bardram *et al.* (2006).

management systems which define tasks based on applications and windows, and is similar to the task representation presented by Card & Henderson Jr (1987) for their "Rooms" system. Further simplifying this definition, Robertson *et al.* (2004) define tasks simply as groups of windows from various applications that are used together.

2.1.2 User Workflow and Workflow Interruptions

Users are susceptible to overload (Adamczyk & Bailey 2004), making user attention and workflow both delicate and difficult to maintain, especially when interruptions occur or work is divided across sessions. Yet, Mark *et al.* (2005) discovered that 57% of "working spheres" (goal-oriented sessions) are interrupted regularly by activities such as co-worker conversations, virus scanner pop-ups and instant messages. Mark *et al.* also found that related interruptions often prove *useful* while unrelated interruptions lead to disruptive shifts in thinking. This echoed findings by Cutrell *et al.* (2000), who investigated the effects of instant messaging interruptions and found unrelated interruptions resulted in much longer overall task completion times. Iqbal & Bailey (2007) developed models to support interruption management by asking users to annotate a task flow animation using three different modal break granularities, gaining good accuracy with content creation tasks. Czerwinski *et al.* (2004) summarise the field by outlining implications for design and suggest designs use lightweight *temporal* cues to better assist the user in visualising both forthcoming tasks and tasks *in limbo*.

Iqbal & Horvitz (2007) reported findings from log analysis that users have an average of 3.74 interruptions per hour. They also reported that the average time spent servicing a diversion was 9 minutes, 33 seconds, indicating (by multiplication of both numbers) that attending to interruptions accounts for close to 60% (or $\sim$ 35 minutes per hour) of measured work time, not including time for mental context shifts. They also found that prior to servicing a diversion users work rate increased dramatically, supporting their hypothesis that users perform state-stabilizing actions before switching to the diversion application. Finally, they found higher task switching rates while servicing a diversion or resuming tasks, and that visibility of windows serves as a reminder to restore them.

Under the premise that high-level goals remain fairly stable, goal-centric task switching interfaces such as Bardram *et al.*'s (2006) implementation of ABC (above), and GroupBar (Smith *et al.* 2003, reviewed in Section 2.4) mandate user grouping of windows and applications as goals (tasks), attempting to allow fast recovery from interruptions by restoring each group as a whole. However, Oulasvirta & Saariluoma (2006) showed that unless task activity can be encoded into long-term working memory, high rates of memory loss can occur across sessions (even tiny interruptions). Scalable Fabric (Robertson *et al.* 2004) uses long-term spatial stability for task groupings which should assist long-term memorability. These interfaces either do not allow windows to belong to multiple tasks or have complicated processes for which to assign task groupings causing cross-goal switching, and add additional management overhead when switching between *intra*-task windows.

2.2 Log-based Empirical Methods

Researchers have performed many longitudinal studies using logging tools to collect data. Mackinlay & Royer (2004) deployed the "Glass Box Analysis environment" software that captures mouse, keyboard and window events in Microsoft Windows. They used this tool to collect data for window thrashing (quick successions of move and resize operations — see Section 2.3.1) and present results as a timeline. This tool is a heavyweight tool capturing vast amounts of data, some unrelated to task switching. Renaud & Gray (2004) present research based around the GRUMPS project collecting low-level usage data such as keystrokes and mouse movements. They present novel strategies to filter data and identify events of interest, and techniques to deal with missing actions. The data collected by low-level tools is a much larger superset to data of interest, and the techniques presented in this paper attempt to provide contexts for filtering the data and eliminating noise from the relevant subset.

2.3 Task, Application and Window Switching Interfaces

Kumar *et al.* (2007) group application switching techniques into three categories, based on the organisation of items within interfaces: *Spatial*, *Temporal* and *Hybrid*. Each interface also exhibits one or more of the following properties depending on how it is used: *window*, *keyboard*, and *external tool*. *Window* refers to using the window to instigate the switch (e.g. by using minimise and maximise or clicking on the window itself). *Keyboard* classifies the use of keys on the keyboard, or a combination of keys (e.g. `Alt+Tab`). Finally, *external tools* are on-screen widgets not including the window itself, such as the taskbar/window list from Windows/GNOME respectively, or the Mac OS X Dock.

Most modern mainstream window systems have different switching interfaces for at least two of these categories, with some interfaces (such as Mac OS X's Exposé[1]) spanning more than one category on their own. It is important to note that an interface with one property may be more useful for some tasks than an interface with another property — it is reasonable, for example, to assume `Alt+Tab` is useful while working with text documents because your hands are already near the keyboard. Likewise, when working with a pair of windows `Alt+Tab` can be used to instantly access the other window, bypassing the dialog (providing no interruptions occur).

[1]Exposé: `http://docs.info.apple.com/article.html?artnum=304786`

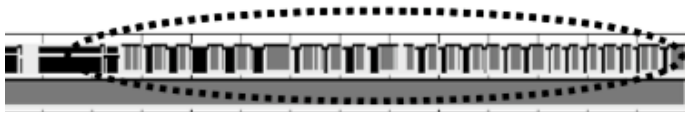


Figure 2.2: Timeline visualisation of logged window activity showing a single work session, with time progressing along the x-axis. Each shade represents a different window, with the small black line at the top of a shade area indicating focus. The circled area shows a period of heavy *window thrashing*—identified by rapid changes in window focus—where the user performs many activate, resize, and move events. Figure taken from Mackinlay & Royer (2004).

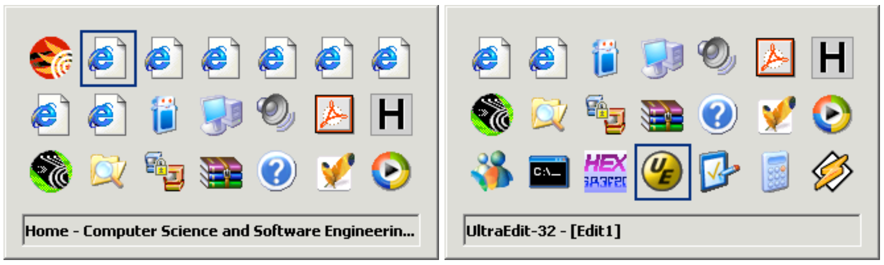
2.3.1 Windows as Widgets

The simplest form of task switching is to use windows themselves as switching tools. To switch focus to a window, a user can to click on the window (or hover, dependant on window manager); clicking directly on a window is a direct mapping to the content required and requires no external mechanics to memorise. The difficulties with this method were touched on in Chapter 1, and first identified by Bannon *et al.* (1983): primarily, limited VDU space leading to occluding windows when more than one or two are open at once. Henderson Jr & Card (1986) quantified this by comparing the size of a number of VDUs to physical workspaces such as desks and tables, calculating that a dining table has the equivalent work space of close to 30 19-inch screens (also Card & Henderson Jr (1987)). Furthermore, Grudin (2001) added additional perspective to this work by identifying that even a large monitor only covers around 10% of the area we can focus on by moving only our eyes. Other researchers have struggled to apply 'messy desk' methodologies successfully with conventional VDUs and have been forced to lower expectations or explore alternative paradigms, often attributing this to display space limitations (MacIntyre *et al.* 2001, Agarawala & Balakrishnan 2006).

Henderson Jr & Card first alluded to the phenomenon of 'window thrashing' in 1986, describing a quick succession of move and resize operations on a group of windows to maintain the desired visibility state, and deemed it to be caused by overlapping windows and small display workspaces. Mackinlay & Royer (2004) analysed activity logs to determine the scale of the phenomenon in modern systems and found frequent occurrences with some thrashing periods nearing minutes in length. Figure 2.2 shows a visualisation from Mackinlay & Royer's logs of one such thrashing period in a single user's window management session. Their findings are likely explained by Iqbal & Horvitz's (2007) observation that the diversity of personal computer use, especially with regard to simultaneous tasks, has grown substantially in the last two decades while the efficiency of switching interfaces has not. By identifying and timing long periods of window thrashing, these studies have provided some evidence to suggest thrashing impedes task switching performance.

In order to combat window thrashing, many research systems (Hutchings *et al.* 2004, Chapuis & Roussel 2005) have focused on the issue of window *visibility*, acknowledging that the need to switch windows is a direct corollary of occlusion caused by limited VDU space. Ideally, all necessary windows would occupy their own space and be available for view without hiding another, analogous to having unlimited desk space. This has lead to one of two common recommendations: a) larger physical display space (through either larger or additional VDUs) (Mackinlay & Royer 2004, Hutchings *et al.* 2004, Hutchings & Stasko 2004a, Robertson *et al.* 2005), or b) using tiling or zooming window management techniques to avoid the problem (Kandogan & Shneiderman 1996, Kandogan & Shneiderman 1997, Hutchings & Stasko 2004b, Chapuis & Roussel 2005). However, these techniques are not without criticism: tiling and zooming techniques do not align well with real-world *desktop* abstractions, larger monitors are scarcely affordable even today, and multiple display use introduces new issues such as gaps formed by screen bezels (Robertson *et al.* 2005) and a general lack of application support (Grudin 2001). Nonetheless, Grudin believes using multiple monitors can yield substantial (though difficult to quantify) efficiency gains.

An older avenue of research focused on two similar concepts: *a*) expanding the working space by using multiple *virtual desktops*; and *b*) maintaining a large movable workspace and using the display as a *viewport*. The latter was first seen in SketchPad (Sutherland 1964, Kay 1987), and is commonly used by image editing programs at close zoom. *Virtual desktops*, influenced by early developments in Smalltalk *project views* (Goldberg & Robson 1983) and Cedar *overlays* (Teitelman 1984), multiply the space available by maintaining hidden workspaces as well as the visible one, allowing the user to control which workspace is visible. Subjective analysis has found gains in using virtual desktops (Henderson Jr & Card 1986, Card & Henderson Jr 1987), and interestingly has found some users prefer them over multiple monitors (Ringel 2003). Virtual desktop implementations are widespread in *X Windows* systems, while Microsoft provide an unsupported extension[2] to Windows to add this functionality.



(a) Dialog after one `tab` press (b) Same dialog after many `tab` presses

Figure 2.3: Two states of the same `Alt+Tab` dialog in Windows XP showing icons for only *some* of the open windows at any one time. A title is displayed for only one icon at any time, and the same icon may appear again for each application window open.

2.3.2 Keyboard Switching Methods

Windows 3.1 introduced the ubiquitous dialog for the key combination `Alt+Tab` (Microsoft 2003), which displays a dialog box on the screen when pressed containing a grid of icons representing all open windows (see Figure 2.3), ordered by a temporally-based *z-order* (Chen 2003). Holding the `Alt` key while pressing `Tab` multiple times iterates through the list, and releasing the `Alt` key brings the window represented by

[2]Virtual Desktop PowerToy: `http://www.microsoft.com/windowsxp/downloads/powertoys/xppowertoys.mspx`

the selected icon to the front and activates it. The title of the selected window only is displayed in a text box at the lower bounds of the dialog. In the worst case (all icons are indistinguishable), `Alt+Tab` is effectively a serial visual search tool; Quinn & Cockburn (2008) found serial menus revealing one item at a time (with mouse movement instead of keystroke selection) are both twice as slow, and twice as error-prone as all other menu types tested for a Zipfian distribution of selection. The list is especially volatile if a user uses more than one switching method, as whenever a window is activated it shifts to the top of the *z-order*, causing it to swap position in the list.

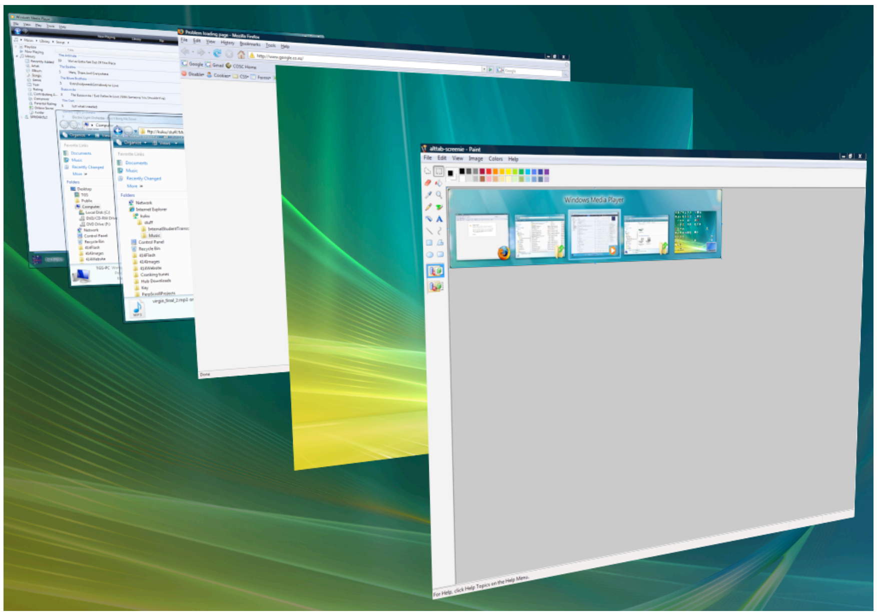
Iqbal & Horvitz (2007) found that users tabbed through on average 7.5 applications before finding their target, indicating long search times. `Alt+Tab` also falters when more than 21 (by default) windows are open, where it only shows a partial list. Figure 2.3 shows two states of the same dialog (i.e. the `Alt` key had not been released). Pressing `Tab` many times scrolls the list to reveal hidden icons (Figure 2.3(b)). Other window managers use different methods: Mac OS X only shows a single icon for each open application in its temporal switcher, and shrinks all icons so the list always fits on-screen. The current release of GNOME increases the size of the displayed dialog box until it eventually overflows the available screen space.

Windows Vista includes an updated `Alt+Tab` dialog that displays a small thumbnail image of each window instead of an icon (Microsoft 2007), and also allows direct selection by mouse (similar to the Mac OS X dialog). However, this technique is not easily discoverable by users familiar with legacy `Alt+Tab`, and also requires users to acquire the mouse with their hand. Figure 2.4(b) shows a screenshot of the new `Alt+Tab` dialog. Vista also includes a new interface called "Flip3D", pictured in Figure 2.4(a), that is mechanically identical to `Alt+Tab` (except that it uses the Windows key instead of `Alt`). Flip3D removes the display of window titles completely and relies solely on the images to provide adequate information entropy for target selection. Both these methods suffer from many of the same problems as legacy `Alt+Tab` and often require visual searching. Indeed, Ramos *et al.* (2006) found *Tumbler*, an earlier interface very similar to Flip3D for layered image manipulation, was slower than competing techniques (including a simple z-ordered list) for selection tasks.

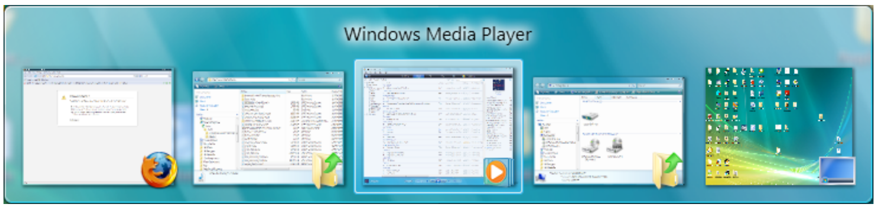
While evaluating the "EyeExposé" interface, Kumar *et al.* (2007) found `Alt+Tab` in Windows XP to be fast when the number of open windows was small. However, they found that its performance scaled worse relative to the other methods evaluated as the number of open windows increased. While we were not able to find other research evaluating the relative performance of `Alt+Tab` to support their findings, they provide anectodal evidence to support their results: `Alt+Tab` is most useful when switching back and forth between two windows but *z-order*ing works against the user (especially with interruptions) by shifting older targets further away.

2.3.3 External Tools and Widgets

External widgets form the largest group of task switching interfaces, involving tools outside the windows themselves to instigate switching. The Taskbar (Windows 95 and subsequent versions) is such an example and belongs to the *hybrid* category. It contains a list of buttons representing each window on the system (including minimised windows), each containing an icon and truncated piece of the window title. Clicking on a button activates the corresponding window and brings it to the fore-



(a) "Flip3D" displays windows in 3D-space with partial z-order occlusion. No titles are displayed, even for the active selection (frontmost).



(b) Vista `Alt+Tab` displaying thumbnails. The active selection has a highlight border around its thumbnail, and a title displayed above the thumbnail list.

Figure 2.4: "Flip3D" and `Alt+Tab` dialogs in Windows Vista. Note that the desktop is also displayed as a window to switch to.

ground at its previous co-ordinates. The list is ordered by first open time and retains absolute spatial stability while all windows on the system remain open. However, if the window list is full and a new window is opened, each button shrinks uniformly in width causing all other buttons to shift left to accommodate the new button. The inverse behaviour occurs when a window is closed: the button is removed, and remaining buttons move left and possibly expand in width to fill the leftover space.

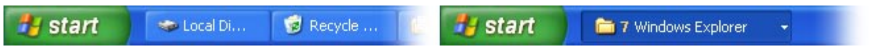
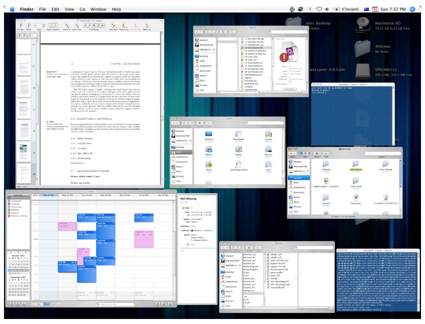


Figure 2.5: Ungrouped window buttons (left) vs. grouped application buttons (right)

Many researchers, including Microsoft themselves, have attempted to improve the Taskbar. Since Windows XP the Taskbar can group by application (Microsoft 2002); when more than a certain threshold of windows belonging to one particular application are open, they collapse into a single *application* button (see Figure 2.5) that


(a) Exposé showing all open windows with eight windows open.

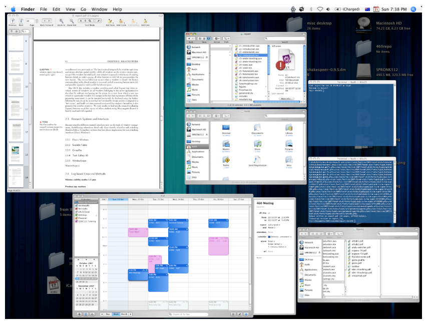

(b) Exposé showing all open windows after closing one *Finder* window.

Figure 2.6: Mac OS X's Exposé in *all windows* mode before and after closing one window. The positions of all other windows changes when a window is closed or moved.

activates a pop-up menu with a list of windows. Both Smith *et al.* (2003) and Robertson *et al.* (2004) claim the grouping by application behaviour confuses many users because application windows may not be related to the same task. Smith *et al.* presented GroupBar (Section 2.4: GroupBar) as a solution, which allows arbitrary user-assigned groups. Robertson *et al.* implemented Scalable Fabric (Section 2.4: Scalable Fabric) as a Taskbar replacement for information workers with larger displays.

Mac OS X includes a window switching tool called Exposé that simultaneously zooms and moves all windows (or all belonging to the active application) to fit without overlapping. The resulting view allows switching to a particular window by clicking on it. It is unique because it possesses all the properties mentioned: *keyboard* with the hot-key then `Tab`, *external tool* invoked by mouse actions, and finally *window* as it adjusts the windows themselves. While we were unable to find specific research evaluating Exposé, *Splatter* (Ramos *et al.* 2006) exhibits similar mechanics to Exposé and was found to be significantly faster than a standard list for selection tasks where a user has rough knowledge of location. Modelling the Fitt's Law time component of this interface suggests that it will be faster than a thumbnailed `Alt+Tab` dialog because the target size for each window is larger (over 300% for large windows).

2.4 Research Systems and Interfaces

"Rooms" Rooms is an early system that strongly resembles today's virtual desktop systems. Henderson Jr & Card (1986) developed Rooms after identifying nine properties task switching interfaces should possess. The system uses a workspace containing a series of interconnected 'rooms' (virtual desktops), each representing a single activity and containing a set of 'engaged tools' (windows). 'Doors' were used as switching widgets to other rooms, with *back* doors to revert to a previous room, assisting task restoration. Rooms also offered users an *overview* mode, which showed a space-filling thumbnail representation of all rooms in alphabetical order.

Data Mountain Data Mountain (Robertson *et al.* 1998) displays a 3D floor plane in perspective view and allows users to place thumbnail images of documents at arbitrary locations with 3D co-ordinates for document management. Against a textual list, Data Mountain performed better for bookmark retrieval in time, error rate and subjective satisfaction. Further investigation (Cockburn & McKenzie 2001, Cockburn & McKenzie 2002, Cockburn 2004) has shown with 2.5D views that Data Mountain's performance was the result of spatial memory and not 3D.

Task Gallery 3D The Task Gallery 3D (Robertson *et al.* 2000) used the visual analogy of an art gallery that users could move through, with a main stage (current work area) and peripheral task displays. Task switching involved swapping the contents of the stage with that of a peripheral task display. Users moved around the gallery with a palette of tools (using a Data Mountain with application icons) in a virtual hand. The Task Gallery 3D was not evaluated against other task switching tools, but users found it enjoyable to use.

GroupBar GroupBar (Smith *et al.* 2003) is an extension of the Windows Taskbar to support *arbitrary* groupings of windows. It adds widgets and drag-and-drop interactions to form groups of window buttons in the Taskbar. It adds a *group tab* to allow automatic arrangement of windows to presets. Task completion times were faster than the Taskbar, and GroupBar gained significantly better satisfaction ratings suggesting users liked the ability to group arbitrarily.

Scalable Fabric Robertson *et al.*'s (2004) Scalable fabric augments a regular workspace with a *periphery* area sized by the user. This area stores spatially consistent minimised windows as thumbnails of the window's last state. These thumbnails can also be grouped arbitrarily to form *tasks*: groups of partially overlapping windows with a name. Scalable Fabric showed no significant performance difference to the Taskbar, though several fixable usability issues were identified in the study.

Activity Bar The Activity Bar (Bardram *et al.* 2006) is the main user interface component from Activity-Based Computing (ABC, Section 2.1.1), providing widgets for each *activity*. Activity Bar supports CSCW tools, zooming, thumbnail pictures for activities and arbitrary grouping where each application and window can belong to more than one activity. Bardram *et al.* (2006) did not evaluate the Activity Bar against another interface but they reported some positive subjective measures.

WindowScape WindowScape (Tashman 2006) is a novel system designed to overcome shortcomings in Groupbar, Scalable Fabric and Kimura (MacIntyre *et al.* 2001). WindowScape allows windows to be represented as 'minatures': thumbnails existing alongside regular windows on the desktop. These minatures remain in the same position regardless of operations performed on the expanded windows, and all minatures can be brought to the top of the z-order with a single command. WindowScape also uses a temporal series of thumbnail screenshots that users can add to a 'favourites' panel. This provides an implicit means to assign windows to multiple task groups (defined by favourite window position and size states). At the time of writing, Tashman has not yet published results of formal evaluations.

2.5 Spatial Memory in Navigable Interfaces

Spatial memory is a powerful tool for user interfaces to exploit. We learn spatial positions very quickly and are able to recall them with great accuracy. Cockburn *et al.*'s (2006) Space-Filling Thumbnails interface for document navigation showed vast improvements in acquisition time with spatial searches (i.e. the user is able to remember position). Furthermore, strong relationships have been found between spatial ability and user performance in multiple different fields including file management and computer gaming. Data Mountain (Robertson *et al.* (1998) - see Section 2.4: Data Mountain) used 3D spatial layouts to display bookmarks with positive results. Cockburn & McKenzie (2001) and other related research showed Data Mountain's performance would have been high with a 2D interface as well, and was due to spatial memory. The Task Gallery 3D (Robertson *et al.* 2000) and its 2D counterpart, Scalable Fabric (Robertson *et al.* 2004) both use consistent positioning to target spatial memory with positive results. Older researchers have even shown a lack of spatial consistency in menu item positioning greatly impedes performance (Teitelbaum & Granda 1983). Finally, Cockburn *et al.* (2007) showed improved spatial learning when interfaces required more effort.

2.6 Summary

In this chapter we have presented definitions for tasks and given an insight into user workflow and interruptions. We presented an overview and classification of mainstream task switching interfaces, and discussed research systems and their approaches to solve problems with task switching processes. Despite vast amounts of research, task switching interfaces remain largely unchanged with the exception of Mac OS X's Exposé and clone interfaces. Flip3D is a visually-improved version of Alt+Tab, requiring the same serial keystroke mechanics to operate. Vista's Alt+Tab is a genuine improvement, but offers no discoverability for much of it's additional functionality. Finally, we have discussed spatial memory in relation to navigable interfaces and it's proven benefits.

Understanding Task Switching 3

This chapter provides insight into tasks and computer use. The following sections detail the design and execution of an empirical study we performed to collect data from real users explicitly detailing their task switching actions with applications and windows. We developed an application, *TrayLog*, to log task switching actions in Windows XP by means of Windows messages and hooks. Following this we analyse this data and gain knowledge of how users manage their applications and documents. Further, we begin to make inferences on how user's concepts of tasks relate to and their actions and tools. Finally, we use the understanding gained to help design new interfaces in order to more efficiently utilise user's cognitive capabilities.

3.1 Experiment Objectives

The aims of this study are to identify trends observed within the participant group that can be used alongside data from other studies (Mackinlay & Royer 2004, Renaud & Gray 2004, and others), and to provide insight into the causality of these trends. We do *not* aim to provide an unabridged resource from which to base concrete theories of task interaction. Objectives are grouped into two sections: *a*) *Quantitative Goals*, concerning quantifiable measurements of user activity, and *b*) *Qualitative Goals*, using log data and subjective responses to help explain and validate our measurements.

3.1.1 Quantitative Goals

We will collect and analyse log data to provide statistical evidence that we can use to answer the following questions:

1. What is the distribution of applications and window switching?
   Are one or two used much more than the rest, or is it uniform?

2. How many applications and windows do users use frequently?
   Do users need to switch to a large or a small number of applications or windows?

3. How do users use current mainstream task switching interfaces?
   Do users predictably neglect any interfaces or favour others?

3.1.2 Qualitative Goals

We will also collect subjective data from participants by means of a questionnaire to be used in conjunction with log data to both validate the log data and answer:

- how much time is spent in each application;
- how frequently applications are launched and closed, or instability is triggered in current interfaces such as the Taskbar;
- how individual work sessions differ;
- what categorisations different users fall into based on their habits;
- what causes these categorisations if they exist; and
- what a reasonable assumption for the number of tasks an interface needs to be concerned with is.

Our questionnaire contained 5-point Likert scales for rating direct mouse clicks, the Taskbar, and `Alt+Tab`, along with space for comments on each. We also included space for comments on how users would improve task switching interfaces, how they usually switch tasks, and what their estimated distribution between the three methods mentioned was.

3.2 Hypothesis

We suspect the distribution for window switches across sessions will follow a power curve and be Zipfian. We expect that within sessions, window switching distribution will follow a power curve, but form a flatter distribution than data from across all sessions. We predict application use will also follow a Zipfian power curve. Finally, we suspect there will be a difference in the task switching methods used by single and dual monitor users.

3.3 Participants and Apparatus

11 Participants (9 male, 2 female) aged 18 through 55, participated in this study. 9 Participants were computer science postgraduates, the remaining participants were computer science staff or office workers. All participants were using Windows XP with Service Pack 2 installed. Six participants (all male) used dual monitor systems, the rest used single monitors. Display resolution ranged between 1280×960 to 1600×1200 for single monitor participants, and 1280×960 to 1680×1050 per monitor for dual monitor participants. Log files gathered ranged from 3 to 20 megabytes and spanned between two and ten weeks of use, averaging 6.4 megabytes and 5.25 weeks.

3.4 Experimental Design and Method

3.4.1 Data Available in Microsoft Windows

Microsoft Windows is a message-based operating system: all actions—from mouse movements to file writing—are performed via the messaging system. Therefore, we can inspect and log user actions by listening to user interface and window management messages which belong to the set prefixed with `WM_`. These messages can be inspected through system *hooks*: chained functions which receieve, process, and pass on a specified set of messages. A programmer can add a hook by instructing Windows to inject executable code from a dynamic link library (DLL) into each running process. Hooks of type `WH_CBT` are executed whenever window actions (activate, focus, minimise etc.) occur, while `WH_KEYBOARD` and `WH_MOUSE` hooks are executed whenever keyboard or mouse actions (press, release, move etc.) occur.

Some applications, especially multi/tabbed document interface applications such as Firefox, do not generate correct message streams or window sets and instead use their own internal systems to operate. However, they still provide data in the form of window titles and positions. Windows APIs exist for traversing the current tree of windows (`GetForegroundWindow()`), inspecting window titles, positions and sizes , and getting the last input event (`GetLastInputInfo()`), all of which are appropriate here. Finally, data from Windows API functions can also augment hooked data by supplying window titles, process IDs for applications, and their executable names (e.g. `notepad.exe`).

3.4.2 Development of TrayLog

TrayLog uses three hooks along with a polling loop to gather user interaction data. While all data could be gathered with greater ease by using a polling loop, we use hooks for efficiency and accuracy. No mouse click can escape a hook message, but it may complete before the next poll. We use a `WH_CBT` hook to determine when windows are switched to and how many windows exist. The `CBT` hook is activated on the following events:

- window creation and destruction,
- window activation,
- focus,
- minimising and maximising,
- movement, and...
- resizing.

We used activation events to indicate a window switch, then use the window handle provided by the hook to query for the window title, executable name and process ID. Unfortunately, the `CBT` hook does not provide details on how the user performed the switching event, nor whether or not a task-switching interface was used (the activation may be the result of closing another window). We determine which switching interface (if any) invoked the switch by installing both a mouse and keyboard hook, flagging relevant keystrokes and mouse actions to deduce the invoker logically. The following rules are processed in this order to determine the interface used:

1. **Alt+Tab**: set a flag when the keyboard hook identifies the key combination. The next switch is marked as `KACTIVATE` (keyboard activation by `Alt+Tab`).
2. **Taskbar**: set a flag when the mouse hook identifies a click. If the click was within the taskbar region, the next switch is marked `TACTIVATE` (taskbar activation).
3. **Direct Mouse Click**: If the click identified by the mouse hook was outside the taskbar region, the next switch is marked `MACTIVATE` for mouse (direct) activation.
4. **No Interface**: Finally, if neither hook is triggered, the next switch is marked `OACTIVATE` (other).

Prior to the `xACTIVATE` message, TrayLog also logs `ALTTAB` and `TBCLICK` (referred to henceforth as *input entries*) for their respective actions so that the noise filter can accurately repair incorrectly-logged actions (see Data Noise Filtering below).

Many MDI applications fail to produce correct message streams (see Section 3.4.1: Data Available in Microsoft Windows), although many adjust the title bar of the parent window when the child document's title changes. TrayLog uses a polling loop every 100ms to store the last sampled title text and log a `PACTIVATE` message whenever the new sampled text does not match the stored text. The `PACTIVATE` log message indicates that the MDI child may have changed and could be a new document.

Each log entry includes a timestamp, an action type (e.g. `ACTIVATE`), the process ID and executable name of the application associated with the action, the window ID (indicated by an integer window "handle"), and the current title of the window. The following text is a small excerpt from a log file (window titles have been truncated):

```
[2007-07-17 22:45:04.171] DESTROYED 2492 787804 msnmsgr.exe MSNMSGRAbsCo...
[2007-07-17 22:45:04.375] OACTIVATE 5120 67780 Opera.exe Tom's Hardware'...
[2007-07-17 22:45:04.375] SETFOCUS 5120 67780 Opera.exe Tom's Hardware's...
[2007-07-17 22:45:04.421] PACTIVATE 5120 67780 Opera.exe Tom's Hardware'...
[2007-07-17 22:45:06.171] MACTIVATE 2492 1574176 msnmsgr.exe Frostastic ...
[2007-07-17 22:45:06.218] PACTIVATE 2492 1574176 msnmsgr.exe Frostastic ...
```

The above log shows the an MSN Messenger conversation closing, causing Opera to activate as the topmost window. Opera automatically sets focus when it receives activation messages and triggers the `SETFOCUS` log. Next, the polling loop detects the active title change and triggers a `PACTIVATE`. Finally, the user activates another MSN conversaion by direct mouse click, resulting in an `MACTIVATE` entry followed by the polled `PACTIVATE` entry.

3.4.3 Data Noise Filtering

The major challenges after collection were to determine what data constituted noise, and to filter it from the final processed data set. We filtered noise using a post-hoc approach, employing PHP script with a set of filter rules. The following paragraphs explain the criteria used for noise filtering and entry repair.

Erroneous `SETFOCUS` Some applications (once again, often MDIs) re-focus each of their UI components upon activation, generating many erroneous `SETFOCUS` log entries. These entries often have little to no identifying information other than the window handle number making them extremely difficult to filter without removing valid data. Ultimately we had to remove `SETFOCUS` entries from our analysis because the scope of these UI components is too wide to filter accurately.

Top Level Window Activation Pollution Applications such as Microsoft Outlook Express use invisible top level windows to control non UI-related routines. In particular, Outlook Express activates these windows routinely whenever it checks a mailbox, generating erraneous log entries in the process. Normally this would generate an `OACTIVATE` log entry and be ignored. However, if the activation occurs between a user performing an activation and the window activation message processing, TrayLog will flag this event as a user activation and incorrectly classify the *actual* action. We did not find an API returning only visible top-level windows that could execute with acceptable speed.

In contrast to widgets, invisible top-level windows usually retain identifiable window titles. For example, Outlook Express' IMAP control windows *always* use the title ``Outlook Express IMAP CFSM Class'' (see Appendix A.1 for the complete title listings). Therefore the filter can remove any log entry belonging to Outlook Express (``msimn.exe'') with such a title, and repair the incorrect entry by using the preceding *input entry*. If no *input entry* is present, the presence of a preceding window destruction event distinguishes between direct click and non-user activation.

Creation and Destruction Not only do some applications activate invisible top-level windows often, many applications also create and destroy them at irregular intervals. Unfortunately, unlike activated top level windows the creation and destruction events can not always be filtered based on window title because the windows often have null or automatically-refreshing window titles. While the filtering system always removes entries with null exe names or window titles (indicating windows that were destroyed before the logger could query for their information), much noise was still present from create/destroy events and thus analysis pertaining to window counts is offered with low confidence only. Fortunately, destruction events often occur in groups, and our basic scenario analysis confirmed our repair function (see previous paragraph) worked correctly by considering these destruction events as a group.

Polled Activations `PACTIVATE` entries occur once for each other `xACTIVATE` message. The final filter stage identified `PACTIVATE` messages pairing with an existing activate message with a timestamp gap less than or equal to 100ms (maximum polling error) and removed them.

Scenario Analysis During the development of the noise filter, we performed a log scenario with known actions, and compared the filtered data to the known data. An excerpt from our log scenario before and after filtering can be seen in Appendix A.2. The final iteration for noise filtering development produced zero errors with the log scenario. It is unfeasable to develop filter profiles for each application, however, although most applications we filter frequently occur in the highest 10 positions in the application switching distribution in Section 3.5.1, such as Mozilla Firefox and Thunderbird.

3.4.4 Session-Marking Techniques

We have many mechanisms at our disposal to mark sessions such as idle timers, explicit start/stop notifications, and window set detection. Despite this, session-marking techniques are difficult to engineer because we must first know what a session consists of before we can utilise marking mechanisms. For example, is a ten minute coffee break or a walk around the office to stretch the legs constitute the end of a session, or is this simply an elongated interruption *within* a session? Are users' sessions independent of interruptions within daily routines? Can sessions span multiple days?

Oulasvirta & Saariluoma (2006) implicitly defining sessions as periods of activity within a working day by indicating a need for long-term memory encoding of tasks (Section 2.1.2). Czerwinski *et al.* (2004) present a near-identical definition of sessions, although they do not mention sessions explicitly instead calling them 'separate activities'. This project uses the same definition for sessions, that is: periods of work within a day, separated by a long interruption. Because the large majority of our participants are from a university, we will define long interruption as anything over one hour, an acceptable time for a lecture or long lunch break. TrayLog marks this definition by using an idle time tracker in the polling loop, recording each time the computer has received no input for one hour, logging a message for various idle periods. We log the messages: `IDLE10S`, `IDLE2M`, `IDLE10M`, and `IDLE1H`, representing ten seconds, two minutes, ten minutes and one hour respectively.

Although we can also use window sets to determine where sessions start and end, our noise filtering showed us that we cannot easily gain high accuracy for window creation and destruction events crucial to window set monitoring. We therefore decided to use idle tracking only to avoid potentially implicit assumptions during analysis.

3.5 Results and Findings

3.5.1 Distributions of Window Switching

We analysed three different distributions using filtered data for both applications and windows, outlined in the three paragraphs below. For all data we analysed only user-invoked switching log entries (`K`, `M`, and `TACTIVATE`), and ignored `P` and `OACTIVATE`.

Application Distribution by Total Number of Switches We measured application distribution across all sessions and found a Zipfian power law distribution for the *n*th frequently targeted application to the total percentage of switches for that application, averaged across all participants. We averaged each participant's first, second, third etc. application across all their sessions, then averaged all participants' applications in frequency order independent of what applications these represented for each participant. Regression analysis showed little variance between our expected distribution and our findings ($R^2 > 0.94$, $y = 70.271x^{-1.7075}$). This distribution is shown

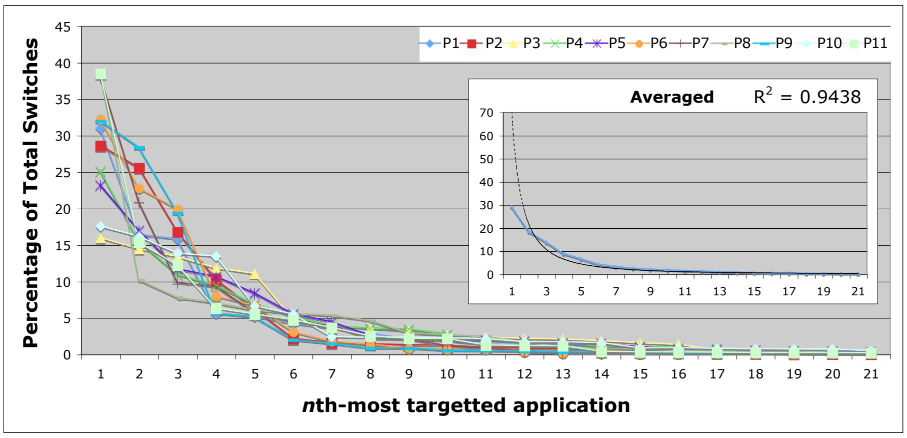


Figure 3.1: Graph illustrating the Zipfian distribution for the percentage of total switches for the 21 most frequently targeted *applications*.

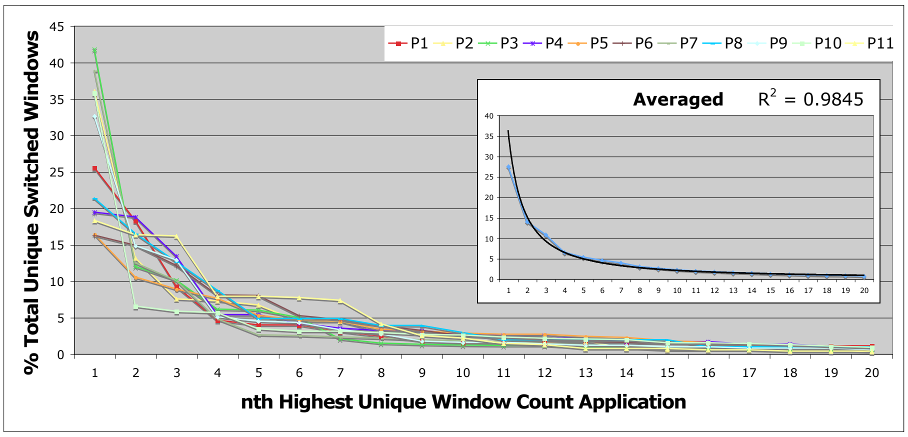


Figure 3.2: Graph illustrating the Zipfian distribution for the percentage of total switches for the 20 most frequently targeted *windows* across all sessions.

by the graph in Figure 3.2, and illustrates that users favour a small set of applications for the majority of their actions. Interestingly, e-mail clients and instant messaging programs *always* featured in the top five targeted applications when present.

All-Sessions Unique Window Distribution We found another Zipfian power law distribution for the *n*th frequently targeted window to the total percentage of all switches to that window, averaged across all participants (illustrated by Figure 3.2). Regression analysis for this distribution showed an even smaller variance than the application switch distribution ($R^2 > 0.98$, $y = 36.145x^{-1.2391}$). This result both confirms our hypothesis that this distribution follows a power curve, and shows that not only do users favour a small number of applications, but a small number of windows in these applications as well.

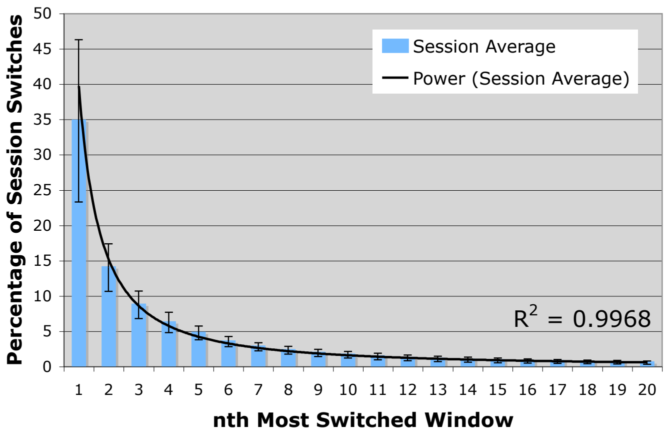


Figure 3.3: Graph showing the average window switching distribution of a session; the first most switched-to window receives far more activations than the second, and so on.

Per-session Unique Window Distribution Using session marking techniques identified in Section 3.4.4 we analysed window switches on a per-session basis and identified a third Zipfian distribution (illustrated in Figure 3.3) with very strong results under regression analysis ($R^2 > 0.99$, $y = 39.763x^{-1.3918}$). This data shows that not only do participants use a small set of windows more than others based on a power curve over the period of a few weeks, they do the same over periods of at most a few hours. We had not expected this distribution to be as pronounced as it is after the previous two distributions were less so.

3.5.2 Task Switching Interface Use — Single vs. Dual Monitors

We found a significant difference in the percentage of both direct mouse clicking and Taskbar clicking to activate windows between single and dual monitor users (Mouse: $F(1,9) = 66.5$, $p < 0.001$, Taskbar: $F(1,9) = 6.0$, $p < 0.05$). After removing the single outlier, our confidence with the mouse and Taskbar results became even higher (Mouse: $F(1,8) = 80.0$, $p < 0.001$, Taskbar: $F(1,8) = 29.741$, $p < 0.005$). This resulted in percentages for single monitor users of 29.7, 65.3, and 5.1 for direct mouse selection, Taskbar and Alt+Tab respectively, and percentages for dual monitor users of 72.8, 23.4, and 3.9, graphed in Figure 3.4. We found no significant difference in `Alt+Tab` use between the two groups, nor did we find any correlation between screen resolution and interface use for either group, a result likely explained by the small range of participant's screen resolutions.

3.5.3 Questionnaire Results

Of the ten participants who returned questionnaire forms, four reported using between 6-15 different windows in a session, and four reported using over 15. Only two users reported using between one and five windows. Our estimate measure from log data supports this - we determined an average of 27 unclosed windows per session. However, these results included non-filterable noise from erroneous log entries

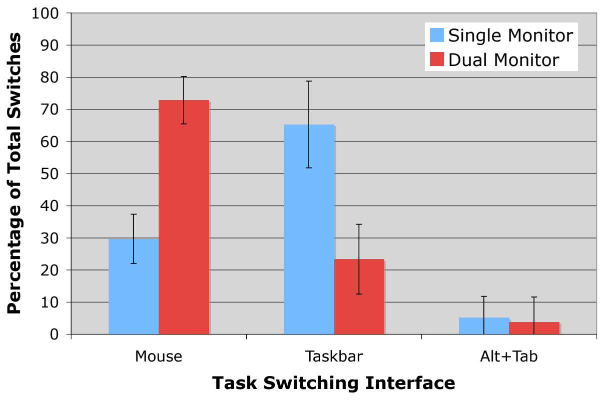


Figure 3.4: Graph showing the percentage of total switches for each task switching interface logged for both single and dual monitor users.

as discussed in Section 3.4.3. Eight users reported switching windows once every few minutes, while two reported switching more than once per minute, consistent with our log findings.

Comparing participants' estimated interface use revealed the same single vs. dual monitor split as the logging data. Single monitor users predicted direct mouse selection for 28.4% of their switches, while dual monitor users predicted direct mouse selection for 62.6% of their switches on average. Both predictions were within eleven percent of the measured results (Section 3.5.2). We calculated the mean of participants' individual differences between their predicted and measured results, as shown in the table below:

	Mouse	**Taskbar**	**Alt+Tab**
Single Monitor	10.6%	13.4%	6%
Dual Monitor	10%	6.4%	5.6%
Overall	10.3%	9.9 %	5.8%

The maximum difference between expected and measured percentages by any participant was 35%, however after contacting the participant we determined TrayLog was working correctly and logging all methods accurately, and that the participant had included switching in MDI applications when considering their interface use percentages. The means presented in the table above show cyclically that both the participants were well aware of their switching interface use, and that TrayLog measured accurate results.

All users echoed similar sentiments regarding each task switching interface. All indicated that direct mouse clicks were only useful when the windows are visible, and six participants also indicated that they would use this technique more often if window occlusion were less prominent. Seven participants found that Taskbar buttons became too small to be useful when high numbers of windows were open. Interestingly, at participants' screen resolutions it takes at least 24 open windows before less than five characters are displayed, possibly indicating that while users only switch to a small number of windows frequently, they may leave a large number of windows open in their workspace. Additionally, five participants indicated that the Taskbar grouping behaviour used by Windows XP solves the problem of reduced text area but at the same time introduces a second click. Only one participant mentioned functionality to group arbitrarily would be useful. Finally, `Alt+Tab` ratings were split in a bipolar fashion between participants: some enjoyed its speed for accessing recent windows, especially when their hands were already at the keyboard. Others complained of an uncomfortable key combination, requiring acquisition of the correct keys, and annoying behaviour when interruptions occur within their task flow.

Finally, seven of the ten questionnaire responses wished for better stability in their task switching interfaces. Some expressed this as annoyance with the juggling behaviour of `Alt+Tab`, while others expressed this explicitly stating they became annoyed with the Taskbar when items moved and resized.

3.6 Discussions and Conclusion

3.6.1 Problems with Experiment

We found only three problems worth noting throughout the experiment. First, we encountered a problem with TrayLog and the Virtual Desktop PowerToy (see Chapter 2). Switching virtual desktops in Windows with the PowerToy requires either an action identified by TrayLog as a Taskbar click, or a key combination not identifiable by TrayLog data. Only one user reported using the PowerToy, and their switching data was consistent with other users in the same group. We attribute this to virtual desktops being used for high level switches in tasks in accordance with their original design (Henderson Jr & Card 1986), whereas more familiar methods such as the Taskbar and `Alt+Tab` were used for *intra*-virtual desktop switching in the same task.

The second problem we discovered was with the Taskbar enhancement and replacement program 'UltraMon'. UltraMon can add a second Taskbar to a dual monitor user's workspace while the Windows API function to determine the Taskbar position returns only the native Windows Taskbar's position. This results in UltraMon Taskbar clicks being registered as `OACTIVATE` system activations and not user activations. Two dual monitor participants reported using the UltraMon application, although both participants' measured percentages were within seven percent of their estimates which we deem an acceptable margin of error. One participant reported using their second monitor as a peripheral monitor for their e-mail inbox and instant messaging which may indicate lower use for the UltraMon Taskbar. Both participants estimated their taskbar use to be low (between 25% to 35%), lowering the possible effect on our switching distributions.

Finally, our third problem was caused by applications we did not know about and could not gain access to for manual filter analysis generating erroneous top-level windows. Manual scrutinising of log data across all participants found in total 320 (of over 30,000 switches) cases where an `OACTIVATE` entry directly followed another. No further investigation was undertaken as we deem less than 1% to be an acceptable error rate for this measure.

3.6.2 Practical Outcomes and Future Work

We found all distributions between percentage of total switches and switch targets were Zipfian, with a 5% threshold landing somewhere between five and eight targets. This is useful knowledge in that we can design interfaces to primarily cater for the small set of applications and windows targeted by 95% of switches. Further investigation could compare the switching distribution to each application and window to the time spent in each application and window and determine any correlation that may exist.

When combined with data from the application distribution, the similarities between our all-sessions and per-session window switching distributions suggest that users stick to a familiar set of applications. In order to gain high frequency for either application or window distributions, the familiar application set needs to be stable. We found for applications that the five percent threshold lies at approximately seven applications. From this we can explore designs for task switching interfaces that utilise spatial memory by assuming a stable set of at least seven applications and assigning them each spatially stable positions.

Both explicit and implicit questionnaire feedback indicated that participants often have many more than five to seven windows open at a time. When paired with the Zipfian distributions, this indicates that seldom-used windows are consuming much of their Taskbar widget area and necessitating visual searches. Users complained of not being able to see enough of the title text in the Taskbar — perhaps the space available for widgets in the Taskbar is inadequate to support user habits. Comparing different sizes of Taskbar may reveal a correlation between performance and Taskbar size at high window counts. Participants also appreciated the grouping functionality of the Taskbar for partitioning space, although they did not like the menu or second click actions. This indicates that task switching interfaces should support grouping, but at the same time *not* require additional visual search steps to acquire targets.

User's favourable responses an feedback for `Alt+Tab` further reinforces the argument that perhaps task switching interfaces are complimentary. Fitting more than a handful of windows on-screen with participant's resolutions seems difficult, and when dual monitor use increases direct clicking with the same per-session window switching distributions, we can determine a need for interfaces that are fast with small numbers of recent items. Once again, speed should *not* come by sacrificing switching capability for high counts and resorting to slow visual search strategies.

Finally, we found how participants described their task switching habits interesting, especially when only one participant wished for arbitrary grouping of switching widgets. Most participants indicated that they switch between windows very frequently (all responded less than once every few minutes). Additionally, research on workflow interruptions has found that up to 60% of working time is spent servicing interruptions (Iqbal & Horvitz 2007). Combining both suggests that arbitrary grouping would indeed serve to burden users more than help them, especially with random interruptions. How would one define a group for "random interruptions"?

To conclude, this study analysed task switching data in the hope of finding practical outcomes and knowledge from real-world usage data. We successfully analysed log data forming clean distributions. Combined with previous research and qualitative feedback we provide clear vectors for improving task switching interfaces.

4 Design and Implementation of a Spatially Consistent Interface

This chapter first forms design considerations based on empirical data and implicit assumptions made in Chapter 3. From there, we use these principles to assist in designing , providing support for decisions made. We present Spatially COnsistent Thumbnails (SCOTZ) as an alternative to mainstream task switching interfaces to better utilise spatial memory in task switching.

4.1 Design Considerations

The previous chapter outlined many of the design considerations we need to consider in designing SCOTZ. We can summarise the four main principles derived from empirical data and qualitative feedback as follows:

1. Users want stable interfaces and dislike targets that move.
2. Interface conditions that require visual search are disliked unanimously.
3. Users want rapid performance with small sets of frequently used items.
4. Users want interfaces to support larger workspaces than their focused set.

While these are *a priori* principles, they are logical when considering the empirical findings. Furthermore, they are good principles to *evaluate* with SCOTZ, and if found true, will allow interfaces that support users' requirements more closely. As you will discover in the following section, SCOTZ supports many of these things by default, that is we used empirical data primarily to fine-tune.

4.2 Interface Model

Spatially Consistent Thumbnail Zones divides the entire screen space into a grid of equally-sized *zones*. Each zone represents a single group of items, and within every zone is a number of thumbnails belonging only to that zone's group. Thumbnails are reduced-size screenshots of a window, and represent the exact window's contents at the time of capture. Importantly, zones are *always* located in the same space and never move. This requires groupings based on stable workspace elements or concepts.

Much research has focused on grouping arbitrarily by task. However, we discovered in Chapter 3 that few users immediately wish for such functionality. We also presented logical reasoning to suggest that adding this functionality may increase user load and harm performance, especially with interruptions. This initial design of SCOTZ groups by application. We have already given evidence of high stability in user's application sets, with 95% of switches targeting less than seven applications. Furthermore, current window systems already group explicitly by applications: they are the next granularity of 'task' from windows and documents. Grouping by application establishes stable elements and stops moving targets (Consideration #1).

SCOTZ addresses the problem of visual search (Consideration #2) in two ways. First, it uses spatially stable areas (zones) for groups (applications), allowing users to remember where their targets are instead of search for their targets. Second, where target location is unknown, SCOTZ uses a divide-and-conquer visual search strategy, similar to the Windows grouped Taskbar, to constrain visual searching to a small subset of all items. Users should have excellent spatial memory for frequently used items and be able to select them quickly (Section 2.5, Consideration #3). Because SCOTZ uses the entire screen area for widget display and facilitates fast visual search strategies where necessary, performance with large window sets should also be high (Consideration #4).

4.2.1 Grid / Zone Interface

The most important unit in SCOTZ is the zone. Each zone needs to be spatially stable: it needs to stay in the same position and retain the same size. The most obvious strategy for accomplishing stability is to use a fixed number of zones. This decision strays from using a dynamic calculation such as the one used in Space-Filling Thumbnails (Cockburn *et al.* 2006) and instead chooses not to accommodate new entries. We chose a square number, using the square root for row and column counts, resulting in possible zone counts of 4, 9, 16, 25, 36 and so on — Figure 4.1 illustrates the possible zone configurations for SCOTZ. Our empirical study suggest any of four, nine or sixteen zones would suffice for 85% to 97% of task switches, and using this arrangement also preserves the display's aspect ratio and avoids looking "out of place". The negative side-effect to this is that SCOTZ will not be usable for 100% of switches, however this can be addressed in many ways that we discuss in Chapter 6.

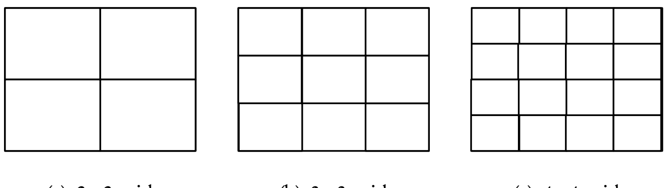

(a) 2×2 grid (b) 3×3 grid (c) 4×4 grid

Figure 4.1: Square number zone configurations for SCOTZ grid.

4.2.2 Internal Zone Space

Because windows are much more volatile than applications and are opened and closed frequently, spacing inside zones must be much more dynamic than the zones themselves. We use a hybrid approach between zone sizing and space-filling techniques: a zone is sized based on the nearest square number to the number of items required. When resize occurs, the current grid is transposed to the upper left coordinates of the new grid, the process illustrated in Figure 4.2.

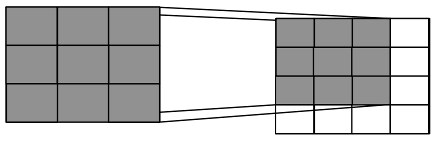


Figure 4.2: Internal zone resizing behaviour moving from a 3×3 zone (left) to a 4×4 zone (right).

This resizing behaviour allows absolute spatial stability inside zones for longer than a dynamic approach by only resizing when absolutely necessary, and by having the new grid condition provide a large number of new slots which must be filled before the next resize event is triggered. Furthermore, transposing the grid instead of simply re-assigning items in a left to right, top to bottom fashion keeps relative spatial stability with the top left of the screen, and with the items themselves

4.2.3 Thumbnails

Thumbnails are the primary means of target identification within the SCOTZ interface. As such, they are resized to fit the divided zone space while preserving their aspect ratio, i.e. tall thumbnails are resized based on their height, leaving gaps at either side. This ensures that users are looking for similarly shaped targets to the windows they are aware of. In order to provide the largest possible target size, these gaps are maintained as part of the target area for selection. Thumbnails display normally at an opacity of 75%, quickly fading in to 100% on mouse hover offering the user hover feedback. A single left mouse click on the target area for a thumbnail activates the corresponding window.

Cockburn *et al.* (2006) mention thumbnail quality as an important factor in their research. SCOTZ has the same reliance on thumbnail quality. Therefore, SCOTZ uses a high quality bicubic resampling from the .net Graphics library to resize thumbnails retaining as much detail as possible. While this is a slow algorithm, we perform the resize as soon as window image data is available and store it in memory—before SCOTZ activation—to make the interface highly responsive.

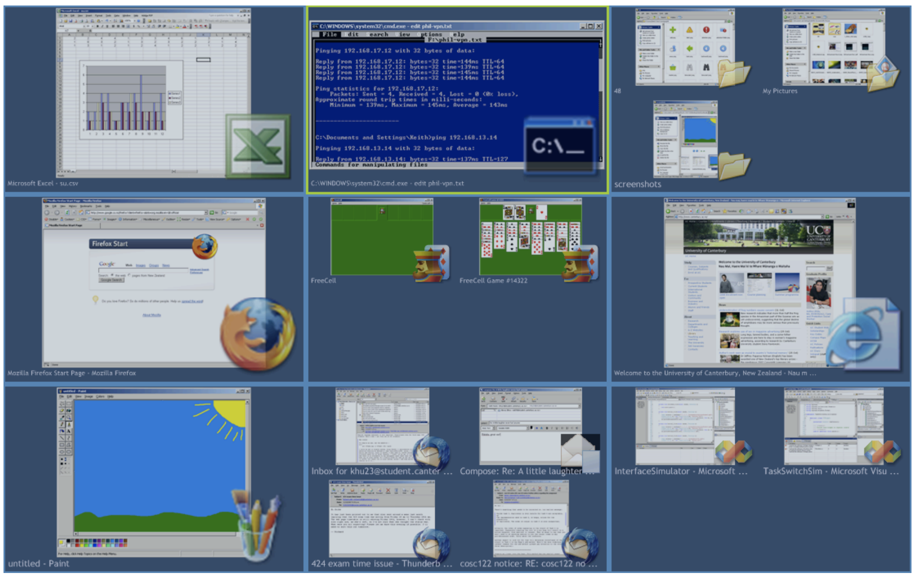
Finally, we overlay the window's icon on top of the thumbnail image at 50% opacity and added a truncated window title to the bottom area of the thumbnail. These provide both additional entropy when using a visual search strategy, and a means to quickly show the zone's associated application. We found 50% opacity to be sufficiently transparent to show both the icon and content underneath with clarity. Icons scaled automatically with thumbnail size to maintain a 15% area overlay.

4.3 Implementation Details

The prototype SCOTZ interface is implemented wholly in C# using .net libraries and makes extensive use of Windows Forms. Our prototype interface requires generated screenshots for window content (although it still generates thumbnails dynamically), as we have not yet implemented functionality to capture a system window.

SCOTZ can be activated by any shortcut combination or mouse button allowed by Windows. Logically, because SCOTZ is a mouse-based switching interface, a mouse button activation is desired. We chose to overload the middle mouse button to activate the SCOTZ grid for our prototype simply because three button mice were readily available. SCOTZ activation would better suit a mouse such as Logitech's MX Revolution with dedicated hardware buttons to suit such tasks.

A screenshot of the final SCOTZ interface using a 2×2 grid is presented below.



Microsoft Excel - su.csv
C:\WINDOWS\system32\cmd.exe - edit phil-vpn.txt
File Edit Search View Options Help
F:\phil-vpn.txt
48
My Pictures
screenshots
Firefox Start
Mozilla Firefox Start Page - Mozilla Firefox
FreeCell
FreeCell Game #14322
Welcome to the University of Canterbury, New Zealand - Nau m ...
untitled - Paint
Inbox for khu23@student.canter ...
Compose: Re: A little laughter ...
424 exam time issue - Thunderb ...
cosc122 notice: RE: cosc122 no ...
InterfaceSimulator - Microsoft ...
TaskSwitchSim - Microsoft Visu ...

```
Pinging 192.168.17.12 with 32 bytes of data:

Reply from 192.168.17.12: bytes=32 time=144ms TTL=64
Reply from 192.168.17.12: bytes=32 time=139ms TTL=64
Reply from 192.168.17.12: bytes=32 time=145ms TTL=64
Reply from 192.168.17.12: bytes=32 time=144ms TTL=64

Ping statistics for 192.168.17.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 139ms, Maximum = 145ms, Average = 143ms


----------------------


C:\Documents and Settings\Keith>ping 192.168.13.14

Pinging 192.168.13.14 with 32 bytes of data:

Reply from 192.168.13.14: bytes=32 time=137ms TTL=127
Commands for manipulating files
```

Figure 4.3: Screenshot of the final SCOTZ interface showing 11 windows open in four different applications with a 2×2 grid.

Evaluation of SCOTZ Interface 5

The following chapter describes an evaluation of the prototype SCOTZ interface introduced in Chapter 4. Using TrayLog data to provide realistic distributions and scenarios for task cueing, we measured acquisition speed for various window densities testing both SCOTZ and three other simulated mainstream interfaces. We found SCOTZ performed fastest under all conditions and gained the best subjective results.

5.1 Experiment Objectives

This experiment aims to determine how quickly SCOTZ allows users to acquire target windows, and how the density of windows affects switching time. Because little published research has been conducted on mainstream interfaces, we also aim to test an application-grouped Taskbar, an ungrouped Taskbar and a serial `Alt+Tab` interface under the same conditions. We also wish to investigate any learning effects from spatial memory when re-acquiring familiar targets. In addition, we are interested in users' subjective satisfaction of the SCOTZ prototype.

5.2 Hypothesis

We predict that SCOTZ will perform faster than the other interfaces for both low and high window density. Further, we predict that SCOTZ will increase in speed faster than the other interfaces as users gain spatial memory of targets and progress to an expert condition. Finally, `Alt+Tab` has been shown to be slower than other interfaces when the number of windows is high (Kumar *et al.* 2007) and we predict a similar outcome.

5.3 Participants and Apparatus

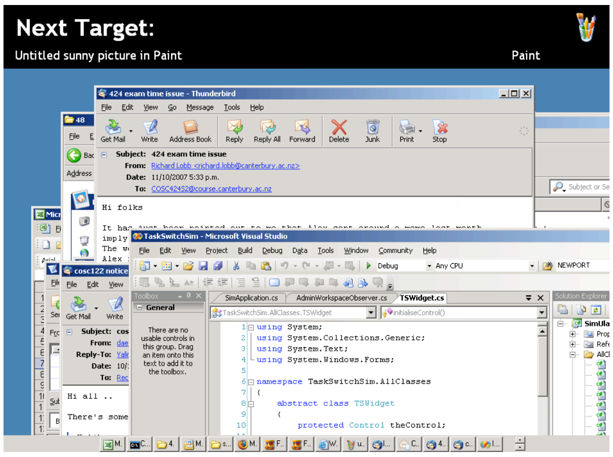
14 participants (2 female and 12 male), aged 19 through 60, participated in this experiment. All participants were familiar with the three mainstream techniques tested. 12 participants indicated they had expert proficiency with computer systems, one moderate and one high.

All participants used 2.13GHz Core 2 Duo PCs with 2GB of RAM running Windows XP Professional. We used single Philips 190S LCD panels running at their native resolution of 1280x1024 with brightness set to 40%. Standard PS/2 keyboards, and Microsoft Intellimouse optical mice connected via USB at 400dpi using Windows XP default control-display settings constituted the input devices.

Most participants were familiar with the mainstream interfaces tested which may influence their performance.

5.4 Tasks and Stimuli

This experiment used a workspace simulator we implemented, including three implemented interfaces using behaviour similar to mainstream task switching UIs. The workspace simulator displayed PNG screenshots as windows, where users could not interact directly with them (e.g. clicking to activate, moving or resizing) or by using auxilary techniques such as `Alt+Esc`. Instead, the task switching interfaces provided were the only methods to switch between windows.


Next Target:
Untitled sunny picture in Paint
Paint

424 exam time issue - Thunderbird
Subject: 424 exam time issue
From: Richard Lobb <richard.lobb@canterbury.ac.nz>
Date: 11/10/2007 5:33 p.m.
To: COSC42452@course.canterbury.ac.nz

Hi folks

TaskSwitchSim - Microsoft Visual Studio

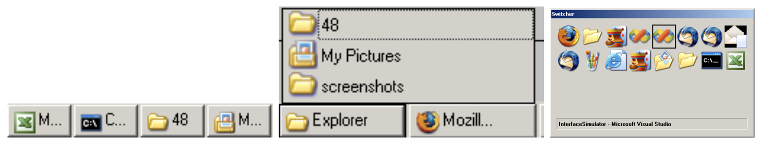
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;

namespace TaskSwitchSim.AllClasses
{
    abstract class TSWidget
    {
        protected Control theControl;
```

Figure 5.1: Screenshot of the workspace simulator showing the task cueing interface with an ungrouped taskbar and a number of open windows.

The workspace simulator uses a task cueing interface in the form of a black bar occupying the top 100 pixels of the screen. Figure 5.1 shows a screenshot of the workspace simulator with a number of windows and an ungrouped taskbar open. A "fuzzy" (e.g. "Your e-mail to Bob" or "Icons 48 pixel Folder") description of the intended target is displayed on the far left hand side of the bar, while the window's icon and application name are displayed on the far right hand side. If a participant makes an incorrect selection, the fuzzy description turns red but allows them to immediately correct their mistake. When a participant selects the correct window, the fuzzy description turns green, and one second elapses before the next task is issued.

Our `Alt+Tab` replacement mimics the `Alt+Tab` dialog in Windows XP with some differences (see Figure 5.2(c). Firstly, it does not automatically centre the list if it does not span the full width. Second, it does not automatically resize the height of the dialog to bring the icon list and window title text closer together. Functionality for both forward (`Alt+Tab`) and backward (`Alt+Shift+Tab`) navigation was available. Releasing `Alt` activates the selected window.

Both Taskbars we implemented did not utilise the areas normally consumed by Start menu, quick launch and the system notification area ("system tray") to provide as close as possible target area to a real Taskbar. The grouping Taskbar (Figure 5.2(b)) used similar functionality to the Windows-style grouping Taskbar. However, we forced a menu to appear even when only one or two windows were open for that application, and *always* presented windows inside a menu. This was done to isolate the two mechanical Taskbar conditions and determine how suitable they are at different window densities. The ungrouped Taskbar (Figure 5.2(a)) emulated Windows' behaviour, including scroll widgets when the list outgrew the widget area (32 window density condition only).



(a) Ungrouped Taskbar (b) Grouped Taskbar (c) `Alt+Tab` Dialog

Figure 5.2: Emulated mainstream task switching interfaces.

5.5 Experimental Design and Method

5.5.1 Interface Familiarisation

All participants were presented with a short three minute demonstration of all interfaces before using any themselves to familiarise them with each interface. Following this presentation, participants were allowed an unlimited number of untimed practice trials with each interface using a dummy set of windows not included in the final evaluation. The training tasks were designed to eliminate learning effects in controls of each interface.

5.5.2 Task Selection

We distributed windows to applications based on the Zipfian distributions determined from the empirical study outlined in Chapter 3 using percentages from the table below. We also used the same percentage set for trial selection (e.g. one window appears in 32% of all trials for one condition).

App/Window #	1	2	3	4	5	6	7	8	9
% Total	32%	21%	16%	10%	6%	6%	3%	3%	3%

The cueing mechanism was implemented by storing 32 references in a collection to the first object, 21 to the second and so on, then using a random number generator to select an index. If the index chosen by the generator contained the current active window, we continued to select windows until this was not the case. Participants were not explicitly told the distribution. The four window and eight window density conditions used a 2×2 SCOTZ grid with four applications, while the 16 and 32 window density conditions used a 3×3 grid with nine applications. These figures were based on the 85% to 97% thresholds reported in Chapter 3.

5.5.3 Experimental Procedure and Design

Participants completed 32 trials for each condition grouped by window density. They were instructed to perform selections as fast as they could, and in the event of an error, immediately try to recover. The first few selections in a workspace for each interface required a visual search as the user had no prior knowledge of the placement of switching widgets for the workspace. Likewise, the fuzzy descriptions for each window had to be processed and window associations determined before switches could be made. We did not record times for the first three trials purposefully to avoid including timing data from initial visual searches. All window density conditions were presented in the same order (8, 4, 32, 16) to avoid maximum fatigue at the 32 window level, and within window density trials we used a latin square configuration for the interface type factor to counter workspace learning effects.

The experimental design is a 4×4 repeated measures analysis of variance (ANOVA) with two factors: *interface type* (SCOTZ, Grouped Taskbar, Ungrouped Taskbar, and Alt+Tab) and *window density* (4, 8, 16, and 32 windows) and one primary dependent variable (acquisition time).

5.6 Qualitative Evaluation

We asked each participant to complete a questionnaire at the end of the experiment in which they rated each of the four techniques using a NASA-TLX scale, ranked each interface relative to one another, and provided comments about all techniques.

5.7 Results

5.7.1 Empirical Results

Figure 5.3 shows the acquisition time results from our quantitative evaluation. A repeated measures ANOVA for interface type and window density showed a significant effect for interface type (F(3,39)=39.459, $p < 0.01$), for window density (F(1,13)=212.852, $p < 0.01$), and interactions between interface type and window density (F(9,117)=19.889, $p < 0.01$). With 14 participants we were unable to find a significant difference between SCOTZ and the ungrouped Taskbar in any of 4, 8 or 16 window densities. However, the mean time for SCOTZ is *always* lowest — no interface was able to beat SCOTZ's mean time for any condition. We found no measurable learning effect across trials, indicating that users learnt spatial positioning extremely quickly — perhaps in the first three untimed tasks.

Regression analysis showed a linear relationship between workspace density and selection time for SCOTZ ($R^2 > 0.97$) and near-linear for the grouped Taskbar ($R^2 > 0.92$). However, the two non-grouping interfaces exhibited explonential relationships

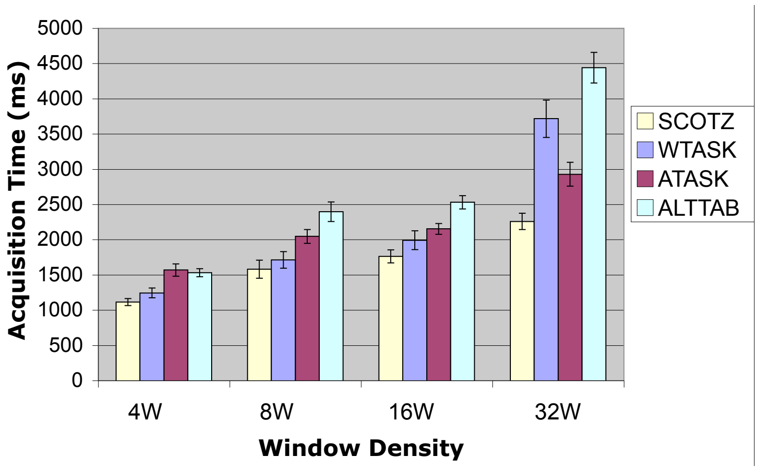


Figure 5.3: Average acquisition times for each interface under each window density condition (n=14) showing SCOTZ performing the fastest in all conditions and scaling much better at 32 windows. ATASK is the Grouped Taskbar, WTASK is the Ungrouped Taskbar.

($R^2 > 0.93$ for `Alt+Tab`, $R^2 > 0.92$ for ungrouped Taskbar). This behaviour can be attributed to the grouping behaviour providing a better strategy for search and selection at high densities.

5.7.2 Subjective Results

Figure 5.4 summarises the results of the NASA-TLX subjetive workload assessment present in the questionnaire. SCOTZ is the clear winner in all categories other than physical load, where it still obtains the highest mean score. Participants unanimously ranked SCOTZ the best of the four interfaces, and had very positive comments enjoying the spatial positioning and stability, speed of the interface, appearance, and ease of use. Users generally ranked thumbnails to be very useful (mean=4.65 out of a maximum of 5), though some users complained the text used with thumbnails was too small and difficult to read. Users also mentioned that in the 32 window density condition, thumbnails of similar windows are difficult to distinguish between.

5.8 Discussion and Conclusions

This section briefly discusses some experimental concerns and biases, and summarises the findings of the prototype evaluation.

5.8.1 Experimental Concerns

We identified three factors which may have introduced bias towards Taskbar interfaces during the experiment. Firstly, a real workspace has dynamic properties: windows are closed and opened frequently. This causes Taskbar buttons to disappear, resize and move. We did not emulate this functionality, and as such Taskbar buttons

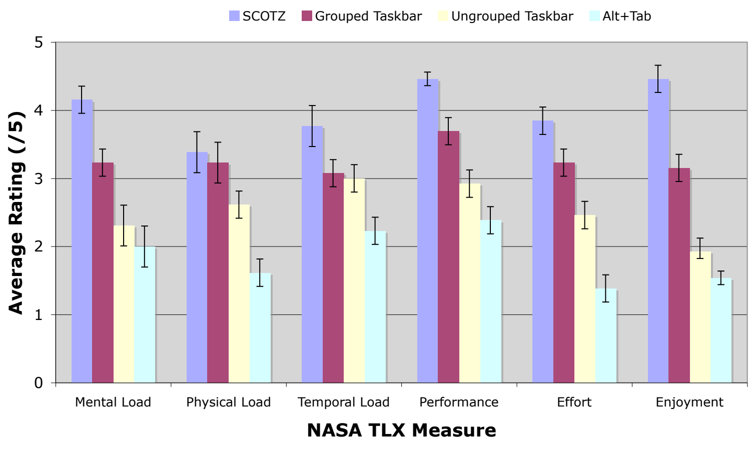


Figure 5.4: NASA-TLX subjective workload assessment results (+/-SE).

gain absolute spatial stability when they otherwise would not. Secondly, because the cueing interface generated windows in their distribution order, the most frequently used windows were always found toward the left. This allowed users to hover in the region and avoid lateral mouse movement time in selections. Finally, users were able to hover their mouse near the Taskbar at all times, waiting for the next task.

Because the SCOTZ interface utilises the full display area, it occluded the task cueing interface when open. While most users appeared to quickly grasp the fuzzy descriptions and had no obvious trouble mapping them to windows and applications in each workspace, one user mentioned forgetting the selection target and being forced to make an unwanted error to see the cueing interface.

The four and eight window conditions may also be reaching the limits of human motor abilities. The Keystroke Level Model (Card *et al.* 1983) indicates that pointing time for a target is between 0.9 and 1.1 seconds, and the time to perform two clicks is 0.4 seconds. Interestingly, the mean time for SCOTZ in the four window condition was 1.15 seconds, the action requiring two clicks and a pointing task. According to KLM, the minimum selection time for this interface should be approximately 1.3 seconds. Likewise, the 8 window condition is also approaching the 1.3 second mean time (SCOTZ 8W mean = 1.58s). We can therefore stress the importance of even a small difference in mean value at this level, even if not statistically significant; the performance of SCOTZ is certainly impressive.

Our emulated `Alt+Tab` dialog did not scroll when more than 21 windows were open, as the standard Windows dialog does. This should not cause a significant performance decrease because the serial nature of the `Alt+Tab` interface necessitates serial visual searches.

5.8.2 Discussions

We found data to support our hypothesis that SCOTZ is faster than other interfaces for all conditions. Although the difference is not significant at the 4, 8 and 16 window densities, SCOTZ wins by a large margin at the 32 window condition. This result supports both design considerations #3 and #4 outlined in Section 4.1, allowing both rapid performance with small sets of frequently used items (SCOTZ performs at least as fast as all other interfaces tested), and support for larger workspaces without sacrificing switching speed for small sets. In addition, subjective analysis showed users unanimously prefer this functionality when compared to other methods.

Interestingly, while SCOTZ requires two mouse click events to activate a window, its nearest competitor (the ungrouped Taskbar) only requires one. Yet, SCOTZ still maintains a lower mean time for selection throughout all conditions. Furthermore, the grouped Taskbar (which also requires two clicks to switch) performed slowly in all except the 32 window condition.

Comparing the results gained for `Alt+Tab` and the ungrouped Taskbar to those of Kumar *et al.* (2007) shows similar trends. For both the four and eight window conditions, they report mean times within 200ms of our results. We also found `Alt+Tab` scales similarly to their findings, with means of approximately 2500ms when approaching 16 window conditions. We report mean times faster on average than Kumar *et al.*, which can be attributed to both using a single monitor for both cueing and selection tasks, and allowing users to hover over the Taskbar. Kumar *et al.* implemented a mandatory movement and selection task for the Taskbar only to stop users hovering directly over the Taskbar at all times (an action not likely to occur in real switching operations). This mandatory post-task selection is one method we could have used to remove the hover bias mentioned in the previous subsection.

To our surprise, no interface exhibited measurable learning effects in our experiment. We take this to indicate that learning occurred extremely quickly, i.e. within the first three trials for each condition. Unfortunately we did not record times for these initial trials, and cannot analyse data from them. If learning did indeed occur quickly, SCOTZ supports the second design consideration in Section 4.1.

5.8.3 Summary

To summarise, we evaluated SCOTZ against four mainstream task switching interfaces and found it to fastest in all conditions measured. SCOTZ also outperformed other interfaces when scaling with workspace size, and received highly favourable NASA-TLX ratings. Finally, SCOTZ received unanimous approval from participants who all ranked it as the best interface of all tested.

Discussions and Conclusions 6

6.1 Implications for Design

Our SCOTZ prototype evaluation supported all four design considerations outlined in Chapter 4. We can now suggest that interfaces utilising stable workspace elements (e.g. applications) in a spatially consistent manner perform well and are preferred by users.

We can also suggest that utilising the distributions discovered in Chapter 3 is a useful metric from which to base task switching interface design on. Our evaluation used distributions from our empirical study for testing with good results, but more importantly used thresholds found to configure the SCOTZ interface with a good number of zones.

6.2 Future Work

SCOTZ can be improved in many different ways: at the moment there are no provisions for supporting an arbitrarily large number of applications. Nor did we cover any strategies better utilise zoning space - some applications, especially those with a small number of switchable windows (e.g. MDI applications), do not require such large target space in the SCOTZ grid. The following section outlines some strategies we have considered for the improvement of SCOTZ, and suggest that they be implemented and tested for performance against the original SCOTZ design, both in formal empirical evaluations and long-term user studies with real systems.

6.2.1 Frequency-based SCOTZ

SCOTZ could benefit greatly by using data collected by an application such as TrayLog to enhance the positioning and sizing of zones and thumbnails. Frequency-based SCOTZ could apply a space division principle such as TreeMaps to resize thumbnail slots according to their use frequency. This way, target sizes could be increased for frequently switched windows or applications, while maintaining relative spatial stability of thumbnails and absolute spatial stability of zones. Further, SCOTZ could continue to use a fixed zone count, and determine which applications to place in which zones based on a frequency metric. This would likely require a technique to catch all windows that did not belong to a zoned application, such as Catchall SCOTZ outlined in the next paragraph.

6.2.2 Catchall SCOTZ

The major obstacle preventing SCOTZ from being a stanalone task switching solution is that to maintain a usable interface, the number of zones will likely be less than the number of applications on a user's system. Catchall SCOTZ assigns a single zone as a 'catch all' zone, where windows who do not belong to a zoned application will be placed. This allows SCOTZ to provide large spatially stable target areas for over 95% of application switches, but also scale to support an unlimited number of possible applications.

6.2.3 Workspace SCOTZ

Workspace SCOTZ is a technique whereby SCOTZ becomes a 'frame' to a workspace as opposed to a layer above. Zones can be distributed evenly around the outside of a workspace, leaving the central area as a working area with large windows, similar to Scalable Fabric (Robertson *et al.* 2004). However, automatic grouping based on application could remove the burden of managing higher-level tasks from users and place it on the system itself. It would be interesting to compare Workspace SCOTZ to Scalable Fabric in a formal evaluation.

6.2.4 Abstract SCOTZ

Abstract SCOTZ is essentially user-managed SCOTZ. Zones now become templates in which users can place either applications or windows. Abstract SCOTZ could allow users to define a set of rules to automatically assign applications and windows to zones, so that Inbox windows always move to one zone while Instant Messenger windows always move to another. Spatial memory has been shown to be improved when users have to use additional effort to acquire positioning tasks (Cockburn *et al.* 2007), and this technique could apply when allowing users to group their own windows. This could allow users to more efficiently serve interruptions, especially if paired with an interruption analysis system similar to those Iqbal & Horvitz (2007) mention.

6.3 Conclusions

We have successfully implemented and conduced an empirical study to characterise some of users' task switching habits, and found very clean distributions for window and applications. We also found a bipolar split between single and dual monitor users, which is another field that needs much observation. SCOTZ, a new interface designed to better support task switching, successfully utilised data from our empirical study and performed favourably in a formal evaluation against three other mainstream task switching interfaces. Not only did it perform favourably in terms of selection time, but it was ranked unanimously by participants as the best interface in the experiment and received many favourable comments with little criticism. Hopefully, more researchers will explore the direction we have taken by using current constructs of applications and windows as a starting point, and utilising spatial memory to minimise visual searches.

Bibliography

Adamczyk, P. D. & Bailey, B. P. (2004). If not now, when?: the effects of interruption at different moments within task execution, *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 271–278.

Agarawala, A. & Balakrishnan, R. (2006). Keepin' It Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen, *ACM CHI* .

Bannon, L., Cypher, A., Greenspan, S. & Monty, M. L. (1983). Evaluation and analysis of users' activity organization, *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, ACM Press, New York, NY, USA, pp. 54–57.

Bardram, J., Bunde-Pedersen, J. & Soegaard, M. (2006). Support for activity-based computing in a personal computing operating system, *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, ACM Press, New York, NY, USA, pp. 211–220.

Bewley, W. L., Roberts, T. L., Schroit, D. & Verplank, W. L. (1983). Human factors testing in the design of Xerox's 8010 'Star' office workstation, *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, ACM Press, New York, NY, USA, pp. 72–77.

Card, S. K. & Henderson Jr, D. (1987). A multiple, virtual-workspace interface to support user task switching, *CHI '87: Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface*, ACM Press, New York, NY, USA, pp. 53–59.

Card, S. K., Moran, T. P. & Newell, A. (1983). *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, New Jersey, USA.

Chapuis, O. & Roussel, N. (2005). Metisse is not a 3D Desktop!, *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, ACM Press, New York, NY, USA, pp. 13–22.

Chen, R. (2003). What is the Alt+Tab order?, *The Old New Thing (Microsoft Blogs)* . `http://blogs.msdn.com/oldnewthing/archive/2003/10/20/55367.aspx`.

Cockburn, A. (2004). Revisiting 2D vs 3D implications on spatial memory, *AUIC '04: Proceedings of the fifth conference on Australasian user interface*, Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 25–31.

Cockburn, A., Gutwin, C. & Alexander, J. (2006). Faster document navigation with space-filling thumbnails, *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, ACM Press, New York, NY, USA, pp. 1–10.

Cockburn, A., Kristensson, P. O., Alexander, J. & Zhai, S. (2007). Hard lessons: effort-inducing interfaces benefit spatial learning, *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, New York, NY, USA, pp. 1571–1580.

Cockburn, A. & McKenzie, B. (2001). 3D or not 3D?: evaluating the effect of the third dimension in a document management system, *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 434–441.

Cockburn, A. & McKenzie, B. (2002). Evaluating the effectiveness of spatial memory in 2D and 3D physical and virtual environments, *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 203–210.

Cutrell, E. B., Czerwinski, M. & Horvitz, E. (2000). Effects of instant messaging interruptions on computing tasks, *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 99–100.

Czerwinski, M., Horvitz, E. & Wilhite, S. (2004). A diary study of task switching and interruptions, *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 175–182.

Goldberg, A. & Robson, D. (1983). *Smalltalk-80: the language and its implementation*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Grudin, J. (2001). Partitioning digital worlds: focal and peripheral awareness in multiple monitor use, *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 458–465.

Harslem, E. & Nelson, L. E. (1982). A retrospective on the development of Star, *ICSE '82: Proceedings of the 6th international conference on Software engineering*, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 377–383.

Henderson Jr, D. & Card, S. K. (1986). Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface, *ACM Trans. Graph.* **5**(3): 211–243.

Hutchings, D. R., Smith, G., Meyers, B., Czerwinski, M. & Robertson, G. (2004). Display space usage and window management operation comparisons between single monitor and multiple monitor users, *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, ACM Press, New York, NY, USA, pp. 32–39.

Hutchings, D. R. & Stasko, J. (2004a). Revisiting display space management: understanding current practice to inform next-generation design, *GI '04: Proceedings of Graphics Interface 2004*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, pp. 127–134.

Hutchings, D. R. & Stasko, J. (2004b). Shrinking window operations for expanding display space, *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, ACM Press, New York, NY, USA, pp. 350–353.

Iqbal, S. T. & Bailey, B. P. (2007). Understanding and developing models for detecting and differentiating breakpoints during interactive tasks, *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 697–706.

Iqbal, S. T. & Horvitz, E. (2007). Disruption and recovery of computing tasks: field study, analysis, and directions, *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 677–686.

Kandogan, E. & Shneiderman, B. (1996). Elastic windows: improved spatial layout and rapid multiple window operations, *AVI '96: Proceedings of the workshop on Advanced visual interfaces*, ACM Press, New York, NY, USA, pp. 29–38.

Kandogan, E. & Shneiderman, B. (1997). Elastic Windows: evaluation of multi-window operations, *CHI '97: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 250–257.

Kay, A. (1987). Doing with images makes symbols: Communicating with computers (Video).

Kumar, M., Paepcke, A. & Winograd, T. (2007). EyeExposé: Switching Applications with Your Eyes. Stanford University GUIDe Research.

Lineback, N. (2006). Xerox Star, *Graphical User Interface Gallery* . `http://toastytech.com/guis/star.html`.

MacIntyre, B., Mynatt, E. D., Voida, S., Hansen, K. M., Tullio, J. & Corso, G. M. (2001). Support for multitasking and background awareness using interactive peripheral displays, *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, ACM Press, New York, NY, USA, pp. 41–50.

Mackinlay, J. D. & Royer, C. (2004). Log-based Longitudinal Study Finds Window Thrashing, *Technical Report 6*, Palo Alto Research Centre. Palo Alto Research Centre.

Mark, G., Gonzalez, V. M. & Harris, J. (2005). No task left behind?: examining the nature of fragmented work, *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 321–330.

Microsoft (2002). Taskbar Grouping Overview.
URL: `http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/win_tray_taskbar_grouping_overview.mspx?mfr=true`

Microsoft (2003). Pressing ALT+TAB to Switch Between Applications, *Microsoft Help and Support* .
URL: *http://support.microsoft.com/kb/79869*

Microsoft (2007). Windows Vista: Features Explained: Windows Flip and Flip 3D.
URL: *http://www.microsoft.com/windows/products/windowsvista/features/details/flip3D.mspx*

Oulasvirta, A. & Saariluoma, P. (2006). Surviving task interruptions: Investigating the implications of long-term working memory theory, *Int. J. Hum.-Comput. Stud.* **64**(10): 941–961.

Quinn, P. & Cockburn, A. (2008). The Effects of Menu Parallelism on Visual Search and Selection, *Proceedings of AUIC'08: Australasian User Interface Conference*. To Appear.

Ramos, G., Robertson, G., Czerwinski, M., Tan, D., Baudisch, P., Hinckley, K. & Agrawala, M. (2006). Tumble! Splat! helping users access and manipulate occluded content in 2D drawings, *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, ACM Press, New York, NY, USA, pp. 428–435.

Renaud, K. & Gray, P. (2004). Making sense of low-level usage data to understand user activities, *SAICSIT '04: Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, South African Institute for Computer Scientists and Information Technologists, , Republic of South Africa, pp. 115–124.

Ringel, M. (2003). When one isn't enough: an analysis of virtual desktop usage strategies and their implications for design, *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 762–763.

Robertson, G., Czerwinski, M., Baudisch, P., Meyers, B., Robbins, D., Smith, G. & Tan, D. (2005). The large-display user experience, *Computer Graphics and Applications, IEEE* **25**(4): 44–51.

Robertson, G., Czerwinski, M., Larson, K., Robbins, D. C., Thiel, D. & van Dantzich, M. (1998). Data mountain: using spatial memory for document management, *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, ACM Press, New York, NY, USA, pp. 153–162.

Robertson, G., Horvitz, E., Czerwinski, M., Baudisch, P., Hutchings, D. R., Meyers, B., Robbins, D. & Smith, G. (2004). Scalable Fabric: Flexible Task Management, *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, ACM Press, New York, NY, USA, pp. 85–89.

Robertson, G., van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Risden, K., Thiel, D. & Gorokhovsky, V. (2000). The Task Gallery: a 3D window manager, *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press, New York, NY, USA, pp. 494–501.

Smith, G., Baudisch, P., Robertson, G., Czerwinski, M., Meyers, B., Robbins, D. & Andrews, D. (2003). GroupBar: The TaskBar Evolved, *Proc. OZCHI* **3**.

Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system, *DAC '64: Proceedings of the SHARE design automation workshop*, ACM Press, New York, NY, USA, pp. 6.329–6.346.

Tashman, C. (2006). WindowScape: a task oriented window manager, *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, pp. 77–80.

Teitelbaum, R. C. & Granda, R. E. (1983). The effects of positional constancy on searching menus for information, *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, pp. 150–153.

Teitelman, W. (1984). A tour through cedar, *ICSE '84: Proceedings of the 7th international conference on Software engineering*, IEEE Press, Piscataway, NJ, USA, pp. 181–195.

APPENDIX A

A.1 TrayLog Noise Filter Source Code

Listing A.1: PHP `Filter()` function used to remove noisy window data before processing.

```php
<?php
 // TrayLog noise filter function - Keith Humm COSC460
 function Filter($datum)
 {
  $exename = trim(strtolower($datum['exe']));
  $title = trim(strtolower($datum['title']));
  $action = $datum['action'];

  switch($exename) {
   // remove any traylog entries - don't need any
   case "traylog.exe": return true; break;
   case "traylog_v1.6_installer.exe": return true; break;

   // ZoneAlarm is almost never activated by a user
   case "zlclient.exe": return true; break;

   // explorer.exe has some strange top-level hWnds (in winXP)
   // that appear with the title FolderView.
   case "explorer.exe":
    switch($title) {
     case "folderview": return true; break;
     case "olechannelwnd": return true; break;
     case "activemovie window": return true; break;
     case "default ime": return true; break;
    } break;

   // periodically, outlook express syncs with imap using a number
   // of top level windows.
   case "msimn.exe": // outlook express
    switch($title) {
     case "outlook express foldersync window class": return true; break;
     case "outlook express imap cfsm class": return true; break;
     case "thorconnwndclass": return true; break;
     case "cicerouiwndframe": return true; break;
     case "directdbnotifywndproc": return true; break;
    } break;

   // Internet explorer 6 uses heaps of random windows...
   case "iexplore.exe":
    switch($title) {
     case "cicmarshalwndaej": return true; break;
     case "dde␣server␣window": return true; break;
     case "cicerouiwndframe": return true; break;
     case "olechannelwnd": return true; break;
     case "sysfader": return true; break;
     case "acrobat␣iehelper": return true; break;
     case "olemainthreadwndname": return true; break;
    } break;

   // ultraedit does a few strange things with dialog boxes...
   case "uedit32.exe":
    switch($title) {
     case "&yes": return true; break;
     case "file␣changed,␣reload␣file?": return true; break;
    } break;

   // opera does some weird stuff with DummyWindowless
   case "opera.exe":
    switch($title) {
     case "dummywindowless": return true; break;
     case "cicmarshalwndmfhb": return true; break;
    } break;

   // MSN messenger has an UnnamedWindow, don't know why!
   case "msnmsgr.exe":
    switch($title) {
     case "msnunnamedwindow": return true; break;
     case "msnmsgrabsconnwindow": return true; break;
    } break;

   // visual studio 2005 generates LOTS of erroneous FOCUS
   case "devenv.exe":
    if($action == "SETFOCUS") return true; break;

   // acrobat reader does all sorts of weird stuff
   case "acrord32info.exe":
    switch($title) {
         case "dde␣server␣window": return true; break;
         case "olemainthreadwndname": return true; break;
        } break;

   // php has erroneous, even as apache module
   case "php.exe":
    switch($title) {
         case "olemainthreadwndname": return true; break;
         case "zend␣timeout␣window": return true; break;
         case "default␣ime": return true; break;
        } break;

   // winamp the popular music player...
   case "winamp.exe":
    switch($title) {
     case "dshow_notif": return true; break;
     case "activemovie␣window": return true; break;
     case "olechannelwnd": return true; break;
     case "cicmarshalwndmhg": return true; break;
    } break;
  }
  return false;
 }
?>
```

A.1. TRAYLOG NOISE FILTER SOURCE CODE

A.2 Scenario Analysis Log

Listing A.2: Before and after filtering log file excerpts from scenario analysis log used to verify noise filter system.

```
—— BEFORE: ——
[2007−...3:28:02.976] ALT+TAB
[2007−...3:28:03.202] ALT+TAB
[2007−...3:28:03:222] KACTIVATE 5235 2394045  msimn.exe Outlook Ex...
[2007−...3:28:03.239] OACTIVATE 5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:03.239] SETFOCUS  5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:04.125] MAXIMISED 5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:04.125] SETFOCUS  5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:04.187] PACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:06.968] DESTROYED 5968 199574   AcroRd32Info.exe DDE...
[2007−...3:28:06.968] DESTROYED 5968 2165314  AcroRd32Info.exe Ole...
[2007−...3:28:07.781] TBCLICK
[2007−...3:28:07.781] TACTIVATE 5235 2394045  msimn.exe Outlook Ex...
[2007−...3:28:07.781] OACTIVATE 2272 8324854  Explorer.EXE TrayLog
[2007−...3:28:07.796] SETFOCUS  2272 10160764 Explorer.EXE FolderView
[2007−...3:28:07.859] PACTIVATE 2272 8324854  Explorer.EXE TrayLog
[2007−...3:28:10.562] OACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:10.562] SETFOCUS  5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:10.562] SETFOCUS  5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:10.593] PACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:10.593] OACTIVATE 5780 5704022  uedit32.exe File cha...
[2007−...3:28:10.671] PACTIVATE 5780 5704022  uedit32.exe File cha...
[2007−...3:28:11.515] SETFOCUS  5780 5704022  uedit32.exe File cha...
[2007−...3:28:11.515] OACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:11.515] SETFOCUS  5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:11.515] SETFOCUS  5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:11.531] PACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:33.878] MACTIVATE 5235 2394421  msimn.exe Outlook Ex...
[2007−...3:28:33.890] OACTIVATE 4488 1508380  MySQLQueryBrowser.ex...
[2007−...3:28:33.953] PACTIVATE 4488 1508380  MySQLQueryBrowser.ex...
[2007−...3:28:41.531] PACTIVATE 4208 1902174  cmd.exe /cygdrive/f/...
[2007−...3:28:51.921] IDLE 10S
[2007−...3:29:06.265] DESTROYED 4672 7604806  php.exe OleMainThrea...
[2007−...3:29:06.265] DESTROYED 4672 2229994  php.exe Zend Timeout...
[2007−...3:29:06.265] DESTROYED 4672 2361152  php.exe Default IME
[2007−...3:29:07.968] OACTIVATE 4488 1508380  MySQLQueryBrowser.ex...
[2007−...3:29:08.015] PACTIVATE 4488 1508380  MySQLQueryBrowser.ex...

—— AFTER: ——
[2007−...3:28:02.976] ALT+TAB
[2007−...3:28:03.202] ALT+TAB
[2007−...3:28:03.239] KACTIVATE 5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:03.239] SETFOCUS  5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:04.125] MAXIMISED 5780 4262772  uedit32.exe C:\Docum...
[2007−...3:28:04.187] PACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:07.781] TBCLICK
[2007−...3:28:07.781] TACTIVATE 2272 8324854  Explorer.EXE TrayLog
[2007−...3:28:10.562] OACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:10.593] PACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:10.593] OACTIVATE 5780 5704022  uedit32.exe File cha...
[2007−...3:28:11.515] OACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:11.531] PACTIVATE 5780 3671212  uedit32.exe UltraEdi...
[2007−...3:28:33.890] MACTIVATE 4488 1508380  MySQLQueryBrowser.ex...
[2007−...3:28:41.531] PACTIVATE 4208 1902174  cmd.exe /cygdrive/f/...
[2007−...3:28:51.921] IDLE 10S
[2007−...3:29:07.968] OACTIVATE 4488 1508380  MySQLQueryBrowser.ex...
```