Understanding and Supporting Window Switching

A thesis
submitted in partial fulfilment
of the requirements for the Degree
of
Doctor of Philosophy
in the
University of Canterbury
by
Susanne Tak

Supervision and Examining Committee

Professor Andy Cockburn	Internal Examiner and Supervisor
Dr Géry Casiez	External Examiner
Dr Cécile Paris	External Examiner
Professor Tim Bell	Co-Supervisor

Department of Computer Science and Software Engineering
University of Canterbury
2011

Abstract

Switching between windows on a computer is a frequent activity, but finding and switching to the target window can be inefficient. This thesis aims to better understand and support window switching. It explores two issues: (1) the lack of knowledge of how people currently interact with and switch between windows and (2) how window switching can be supported better.

Having a good understanding of how users interact with windows is important for informing the design of new and improved window management tools. However, there have been relatively few empirical studies of window manipulation on commonly used operating systems, and those that do exist may no longer reflect current use. To address this lack of knowledge a three week log-based longitudinal study of window use by 25 participants was conducted using the custom-made tool *PyLogger*, which recorded actual window switching behaviour.

However, the analysis of longitudinal log data, such as the data gathered by *PyLogger*, is problematic as it is difficult to extract meaningful characterisations. Therefore, this thesis also presents a visualisation tool called *Window Watcher* that assists understanding and interpreting the low level event logs of window use generated by *PyLogger*. *Window Watcher*'s design objectives are described, and examples demonstrate the ways that it summarises and elucidates window use.

The results of the *PyLogger* study provide an empirical characterisation of interaction with windows, and results include the following: (1) the participants had fewer windows open and visible than in previous studies; (2) window switching is a frequent activity; (3) several findings related to specific window switching tools, including that acquiring a particular window by navigating through application-grouped items on the Taskbar is slow, and that Alt+Tab is seldom used for retrieving anything other than the most recently used window; (4) an updated classification of stereotypical window management styles (pilers, maximisers, near maximisers, and splatterers); and (5) there are strong window and application revisitation patterns. Finally, implications of the results of the log study for the design of window switching tools are discussed.

The findings from the *PyLogger* study led to the development of a new window switcher called SCOTZ (for Spatially Consistent Thumbnail Zones). SCOTZ is a window switching interface which shows all windows grouped by application and allocates more space to the most frequently revisited applications. The two design objectives of SCOTZ are (1) to provide a spatially stable layout of applications and windows, and (2) to support revisitation to recently and frequently used windows. Additional design objectives are to support various display sizes, to support both keyboard and mouse input, to provide possibilities for application launching and to provide options for end-user customisation.

The design and features of SCOTZ are described, followed by theoretical and empirical validation of its underlying design principles. Findings include that (1) spatially stable layouts allow for faster acquisition of targets than recency and random layouts, (2) the instability inevitably caused by size morphing does not severely impact user performance, (3) size morphing leads to an overall performance advantage because of the Fitts' Law targetting time advantage of increased target size, and (4) size morphing facilitates finding items because of *guided search*. Also, findings from an empirical study demonstrate that SCOTZ yields performance and preference benefits over existing window switching tools.

Finally, as SCOTZ employs a *treemap* algorithm to generate the layout of the application zones the suitability of various treemap algorithms for the purpose of SCOTZ is explored, particularly in terms of spatial stability. In previous work, many different treemap algorithms have been proposed, often with the aim being to optimise performance across several criteria, including spatial stability. However, none of the existing treemaps are stable when data updates, *and* when items are added/deleted, *and* when many changes have taken place (i.e., the *cumulative* effect of data changes). Therefore, this thesis introduces the novel 'Hilbert' and 'Moore' treemap algorithms, which are designed to achieve high spatial stability. Their performance is theoretically assessed in comparison to other treemaps by using various metrics, including a novel 'location drift' metric to better capture spatial stability than the commonly used 'distance change' metric. The theoretical evaluation demonstrates that Hilbert and Moore treemaps perform well across all stability metrics. An empirical study examines the validity and usefulness of the location drift metric, showing that location drift can explain some effects on user performance that distance change alone can not.

Technical Acknowledgements

This research is a continuation of the Honours project by Keith Humm (University of Canterbury, 2007). Andy Cockburn first came up with the design idea for SCOTZ and inspired this research, and SCOTZ was programmed by Joey Scarr. The *Tobii T60* eye tracker (Chapter 7) was provided by Kari-Jouko Räihä.

Research Ethics

All studies and experiments described in this thesis were undertaken with the consent of each participant, and participants had the right to withdraw from any study at any point in time, including withdrawal of any information provided. The work was reviewed and approved by the University of Canterbury Human Ethics Committee as part of Prof. Cockburn's Marsden Project 07-UOC-013.

Table of Contents

List of Tables

List of Figures

Chapter I

Introduction

In recent years the computer has become a ubiquitous and, for many people, essential piece of technology. It is no longer used by only an elite group of 'tech-savvy' users, and many modern computers are easy to use with little to no training.

The *user interface* provides an understandable mechanism through which the user can interact with the computer, and it 'shields' the user from having to deal with the complex underlying hardware. User interfaces have evolved to a high level of sophistication. This is in contrast to the early days of desktop computing, when the main interaction mechanism was the command line. In 1981, the Xerox Star was the first commercially available computer that had a Graphical User Interface (GUI), as opposed to the (typed) command line. The particular style of interaction employed on the Xerox Star is often referred to as WIMP, for Windows, Icons, Menus and Pointing. Windows allow multiple applications and documents to be open concurrently. Put formally, windows are transient containers that hold an application, document, or any (hierarchical) combination of the two, and windows can be opened, closed, moved and resized. Windows remain a very prominent element in contemporary user interfaces.

The topic of this thesis is the interaction with these *windows*, and in particular switching between them. In their daily computer use, users constantly switch between windows (i.e., make a different window the *focal* window) as they navigate between documents and applications. Previous work has demonstrated that window switching is a frequent activity: in a three week longitudinal study, Hutchings and Stasko (2004a) found a mean window activation time of 20.9 seconds, and a median of a mere 3.77 seconds. Also, a study by Gaylin (1986) showed that window switching activities are far more frequent than window creation, deletion, or geometry management.

Unfortunately, finding and switching to the target window can be inefficient. Direct selection of the target window (by clicking on it with the mouse) is often

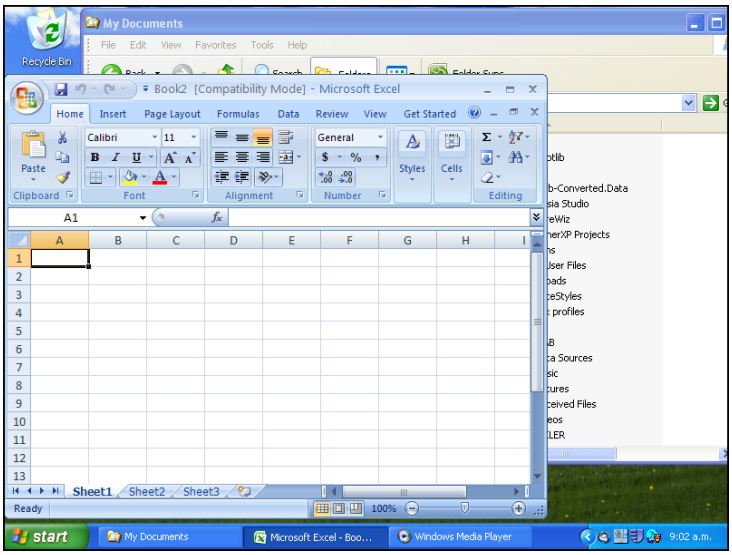


Figure 1.1: Three windows in Windows XP.

not possible because the target window is not visible. In this case, or when the user fails to notice that the target window is visible, a *window switching interface* needs to be used. For example, see Figure 1.1: the Windows Media Player window can not be switched to by clicking on the window itself; a window switching interface such as the Windows Taskbar needs to be used to switch to this window. Even if direct selection of the target window is possible it might not be desirable, e.g., on large displays when the window is located far from the mouse cursor.

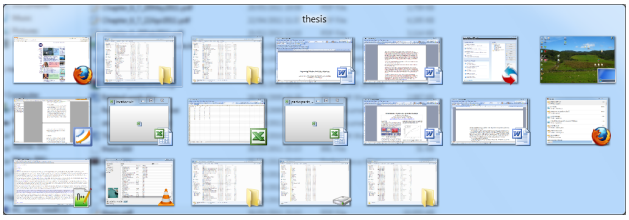


Figure 1.2: An overcrowded Alt+Tab window in Windows 7.

Many common window switching interfaces do not assist the user in finding and selecting the target window. For example, window titles can become too truncated to be readable if a window switching interface is very full. Also, small window thumbnail previews might not be distinctive enough to differentiate between similar-looking windows (for example, see Figure 1.2). Another problem is

that many window switching interfaces are not spatially stable, i.e., the locations of window representations in the switching interface can change or are difficult to anticipate. This instability hinders the development of spatial memory for item locations. When the user is not able to learn item locations due to an ever-changing layout he/she will have to resort to a visual search through all the items, which is often slow. For example, in the Windows Alt+Tab window (see Figure 1.2) the positioning of windows, or the window representations, can be different from switch to switch, hindering the development of spatial memory.

The goal of the current research is two-fold: (1) to further explore and characterise how people switch between and organise windows and (2) to develop a new window switching interface that specifically tries to exploit the characteristics of window use, as well as provide a layout that is spatially stable. As window switching is such a common task, any small improvement of the speed of the interaction can lead to large cumulative gains.

1.1 Research domain

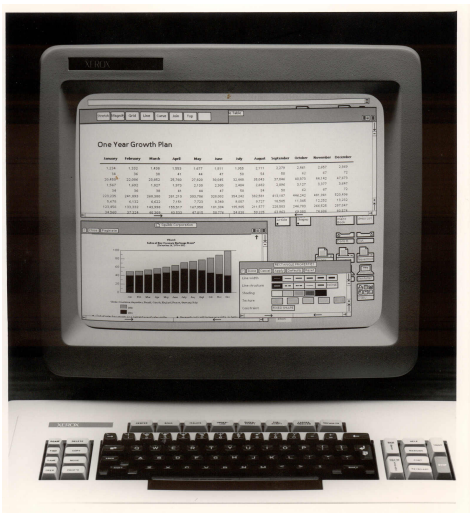
The research in this thesis focuses on window switching on 'regular' one- or two-monitor desktop settings. A wide variety of research and commercial window switching tools are examined in Chapter 2, but the thesis particularly focuses on the commercially dominant Microsoft Windows operating system.

1.2 Current situation

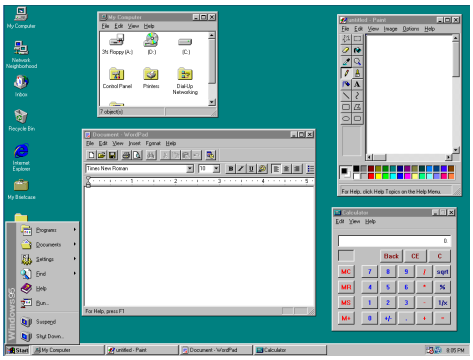
Unsurprisingly, many researchers have identified the need for better window organisation and switching tools (e.g., Bardram et al., 2006; Beaudouin-Lafon, 2001; Bernstein et al., 2008; Chapuis and Roussel, 2007; Fono and Vertegaal, 2005; Henderson and Card, 1986; Kandogan and Shneiderman, 1996; Kumar et al., 2007; Oliver et al., 2008, 2006; Robertson et al., 2004, 2000; Smith et al., 2003; Tashman, 2006; Xu and Casiez, 2010, reviewed later).

Recent commercial developments in window switching interfaces have for a large part focused on richer presentations of windows, such as the addition of window previews to the Windows Taskbar and Alt+Tab in Windows Vista. However, apart from these richer graphical depictions, such as the use of colour, drop

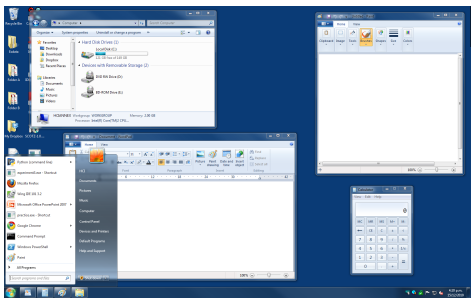
shadows and transparency the computer desktop looks similar to the first WIMP GUIs (van Dam, 1997) (see Figure 1.3). Also, although the visual fidelity of windowing systems has been refined, the fundamental tools for window management have changed little over decades of use.



(a) Xerox Star



(b) Microsoft Windows 95



(c) Microsoft Windows 7

Figure 1.3: (a) The 1981 Xerox Star (Source: DigiBarn Computer Museum, `http://www.digibarn.com/stories/desktop-history/index.html`), (b) the 1995 Microsoft Windows 95 operating system and (c) the 2009 Microsoft Windows 7 operating system.

Having a good understanding of how people use current window switching tools is critical for effective design refinement. However, there are few recent studies of what users actually do. Hutchings et al. (2004) and Hutchings and Stasko (2004a) studied window management and display organisation, respectively, but as these studies are several years old the results may no longer reflect current use

as (1) the size and resolution of computer displays has changed, (2) multi-monitor environments have become more widespread, and (3) tabbed application windows and multi-document interfaces (which allow many documents to be encapsulated within a single window) have become more prevalent.

Also, there are very few studies concerning the relative efficiency and effectiveness of window switching tools. To date, analysis of current commercial methods for switching between windows is often anecdotal and sometimes conflicting. For example, some label the ordering of window representations in the Alt+Tab window as "very effective" (de Chiara et al., 2004, p. 366), but Alt+Tabbing is also considered to be "tedious" (Grudin, 2001, p. 462).

1.3 Research objectives

The current research addresses two problems: (1) the lack of knowledge of how people use, organise and switch between windows and (2) the inadequacy of current window switching interfaces.

Knowing the needs and requirements of a user is an important determinant for successful interface design. However, there is surprisingly little knowledge about how users coordinate their work across windows, how they organise windows on the screen and how they switch between windows. Without this knowledge it is hard to redesign and improve window switching interfaces. Therefore, one of the objectives of this research is to learn more about how people interact with windows.

The second objective is to develop a window switching interface which addresses the needs of the user better than existing window switching interfaces. This thesis identifies two problems with current window switching interfaces: a lack of stability of window locations and a lack of connection to actual window switching behaviour. The premise of this research is that window switching can be made more efficient and effective by (1) providing a more stable organisation of windows in the window switching interface so users can use quick gesture-like target acquisitions and (2) basing the design of the window switching interface on actual window switching behaviour.

1.4 Contributions

Longitudinal studies of window use are rare and those that do exist are already several years old, and the results therefore most probably outdated. Therefore, this thesis presents a longitudinal log study of window use. The analysis focuses on various aspects, including revisitation patterns to windows and applications, the methods people use to switch between windows, and how people organise windows on their screen, including moving and resizing actions. Overall, this study revealed several interesting window use patterns and behaviours, such as strong revisitation, which inspired the design of the novel window switching tool *SCOTZ* (Spatially Consistent Thumbnail Zones). The data is also potentially useful for other developers of tools to support window use.

The log study was paired with the development of a visualisation tool called *Window Watcher*. *Window Watcher* supports the extraction of meaningful characterisations from the data. *Window Watcher*'s design objectives are generalisable to other domains.

SCOTZ is a novel window switching interface, and its design draws on an empirically derived characterisation of window switching as well as exploiting human spatial memory. The layout of SCOTZ uses treemaps, and this thesis presents several studies evaluating the performance benefits of layouts that employ treemaps. These experiments have led to general insights into the effects of size morphing and (in)stability of visual layouts. Also, additional experiments support and validate the underlying principles of SCOTZ and examine the efficiency and effectiveness of two major commercial window switching interfaces (Windows Taskbar and Windows Alt+Tab).

Finally, as SCOTZ uses a treemap for the layout of the elements in the interface, this thesis proposes two new treemap algorithms. These algorithms are described in detail and can therefore be used by other researchers and developers.

1.5 Publications related to this thesis

Some material in this thesis has previously appeared in peer-reviewed publications. An overview of all related publications is given here, as well as the chapters from which material is used.

- Tak, S., & Cockburn, A. (under revision). Enhanced spatial stability with Hilbert and Moore treemaps. (Chapter 8)

- Tak, S., Scarr, J., Gutwin, C., & Cockburn, A. (2011). Supporting Window Switching With Spatially Consistent Thumbnail Zones: Design and Evaluation. *Proceedings of INTERACT '11, Part I*. Lisbon, Portugal, 5-9 September, pp.331-347. (Chapters 6 and 7)

- Tak, S., & Cockburn, A. (2010). Improved Window Switching Interfaces. *Extended Abstracts of the 28th International Conference on Human Factors in Computing Systems (CHI 2010)*. Atlanta, Georgia, USA, 10-15 April, pp. 2915-2918. (Chapters 6 and 7)

- Tak, S., & Cockburn, A. (2009). Window Watcher: A Visualisation Tool for Understanding Windowing Activities. *Proceedings of OzCHI '09*. Melbourne, Australia, 23-27 November, pp. 241-248. (Chapter 4)

- Tak, S., & Cockburn, A. (2009). Performance of spiral and squarified treemaps. Presented at *"Human Aspects of Visualization" workshop at INTERACT '09*. Uppsala, Sweden, 24-28 August. (Chapter 8)

- Tak, S., Cockburn, A., Humm, K., Ahlström, D., Gutwin, C., & Scarr, J. (2009). Improving Window Switching Interfaces. *Proceedings of INTERACT '09*. Uppsala, Sweden, 24-28 August, pp. 187-200. (Chapter 7)

- Tak, S. (2009). Improving Task Switching Interfaces. *New Zealand Computer Science Research Student Conference '09*. Auckland, New Zealand, 6-9 April. (Chapter 7)

Chapter II

Window Management Tools

This chapter presents an overview of the various tools that are available for window management (including window switching and organisation), both the ones that are available in common operating systems and research tools.

2.1 Commercial window switching tools

Modern operating systems have several tools to support window switching included by default. The two standard tools in Microsoft Windows are the Taskbar and Alt+Tab, while Mac OS has the Dock, Command+Tab, and Exposé. This section describes these tools, as well as empirical research related to them.

2.1.1 Microsoft Windows

The Taskbar

The Windows Taskbar is a narrow strip at the bottom of the screen (this is the default position, the position can be modified by the user) and was first introduced in the Microsoft Windows 95 operating system. The Windows Taskbar comprises of a button to access the Start Menu, buttons for launching applications ('Quick Launch', no longer present in Windows 7), buttons for all open windows and the 'Notification Area' (mainly used for status information). For convenience, when 'Taskbar' is used in this thesis it is meant to denote only the part of the Taskbar with the window buttons.

On early versions of the Windows Taskbar (Windows 95 to Windows 2000) the order of the Taskbar buttons was the same as the order in which the respective windows were opened, filling up the Taskbar from left to right. In Microsoft Windows XP *Taskbar button grouping* was introduced. Taskbar button grouping causes windows of the same application to always be next to each other on the Taskbar (see Figure 2.1). Also, when available space on the grouped Taskbar

becomes scarce windows of the same application are *collapsed* under one application button (see Figure 2.2). This collapsing means that two clicks are required to switch to a window rather than one: a click on the Taskbar button for the application group first, then selecting the window.

Figure 2.1: The Windows XP Taskbar.

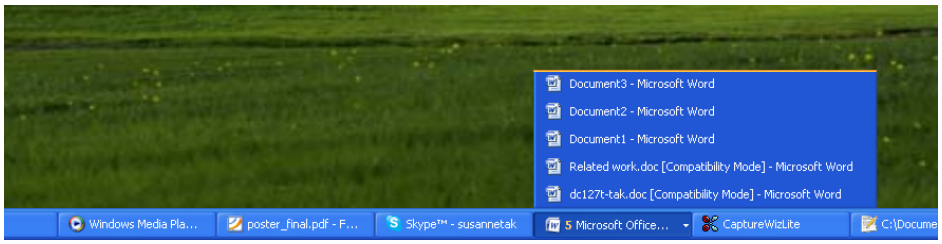


Figure 2.2: A *collapsed* button on the Windows XP Taskbar. The number (5) next to the application title indicates how many windows are open for the application.

With the introduction of Microsoft Windows Vista thumbnail previews of the windows were added to the Taskbar (see Figure 2.3). This can be expected to benefit user performance, as people are better at recalling visual elements of files than they are at recalling titles or file names (Blanc-Brude and Scapin, 2007).

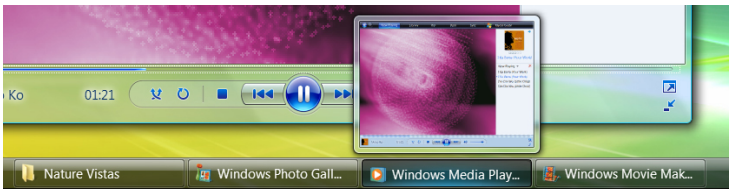


Figure 2.3: The Windows Vista Taskbar (Source: `http://www.microsoft.com`). Hovering over a Taskbar button reveals a thumbnail preview of the associated window.

The Microsoft Windows 7 Taskbar does not show application/window titles (see Figure 2.4). The Windows 7 Taskbar only shows application icons, and these icons can optionally be 'pinned' to the Taskbar so they are always in the same location. Also, pinned application icons remain on the Taskbar even when there

are no windows open for that application. In this case, clicking on the Taskbar icon will launch the application. If there are several windows open for an application these can be reached via a fanned out sub-menu which shows the window titles and thumbnail previews of the window content (see Figure 2.4).

Finally, it is important to note that the Windows Taskbar is only shown on one monitor (the primary monitor) when a multi-monitor setup is used. Czerwinski et al. (2003) propose that the Windows Taskbar should, ideally, be stretched across monitors on multi-monitor setups.

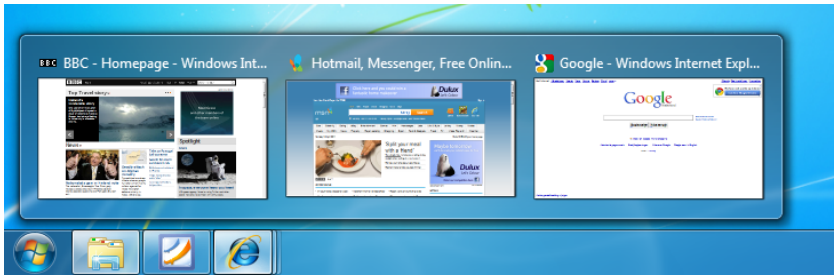


Figure 2.4: The Windows 7 Taskbar. Only application icons are shown, no application/window names. Hovering over or clicking on an icon reveals thumbnail previews and titles of all windows associated with the application.

Alt+Tab

In Microsoft Windows 3.1 the 'Alt+Tab' method for window switching was introduced. Alt+Tab is a key combination for window switching: by pressing and holding down the 'Alt' key and (repeatedly) pressing the 'Tab' key the user can sequentially step through a list of available windows[1]. Releasing Alt+Tab switches to the window selected at that moment. With the introduction of the Microsoft Windows Vista operating system the target window can be selected by using the mouse to click on the desired window as well.

The ordering of the list of windows (or their representations) in the Alt+Tab window is based on the z-ordering of windows. Z-order is similar to recency order,

[1] From the Microsoft Windows 95 operating system and onward pressing Alt+Tab reveals a list of *all* windows. In the early versions of Alt+Tab (in Microsoft Windows 3.1 and 3.11) pressing Alt+Tab revealed a small rectangular window in the middle of the screen with only the icon and the title of the *next* window in the z-ordering.

and is the depth order of windows. A window that is on top of another window is relatively higher up in the z-ordering. No two windows are on the same position of the z-ordering; one window is always higher or lower than the other. When a window is minimised it is sent to the bottom of the z-ordering. Windows Alt+Tab cycles through the windows according to their position in the z-ordering; from front (top) to back (bottom). By using Shift+Alt+Tab the user can cycle through the windows in reverse order. Since the introduction of the Microsoft Windows Vista operating system only the first six windows in the Alt+Tab representation are shown by z-order, the others are listed alphabetically by application name.[2]

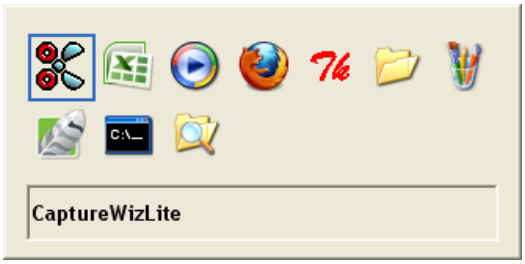


Figure 2.5: The Alt+Tab window in Windows XP.

From Microsoft Windows 95 to Windows XP Alt+Tab showed icons only and the window title was only shown for the currently selected window (see Figure 2.5). Since Microsoft Windows Vista the Alt+Tab window also shows small window previews (see Figure 2.6), but the window title is still only shown for the currently selected window.

Figure 2.6: The Alt+Tab window in Windows 7.

Finally, Microsoft Windows Vista also introduced an enriched version of Alt+Tab called 'Flip 3D' (see Figure 2.7). The functionality of Flip 3D is identical to that

[2] `http://blogs.msdn.com/b/oldnewthing/archive/2008/07/01/8673981.aspx`, accessed 8 November 2010

of Alt+Tab (which is sometimes referred to as 'Flip'), but the difference is that the windows are shown in a full-screen stacked 3D view.

An advantage of Alt+Tab is that it allows for very quick retrieval of the most recently used window. However, a disadvantage is that sorting windows by z-order is spatially unstable: the ordering of the window representations will be different from switch to switch if the z-ordering of the windows changes. Also, it is unclear how well users understand and can anticipate z-order.

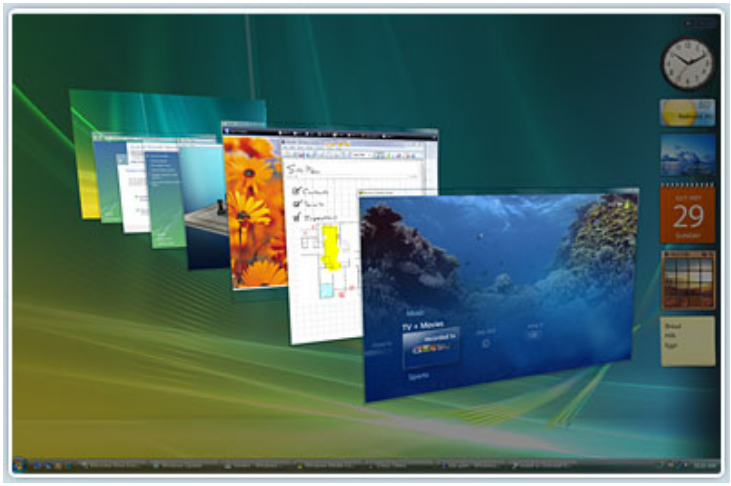


Figure 2.7: Flip 3D in Windows Vista (Source: `http://www.microsoft.com`).

2.1.2 Mac OS

The Dock

The Mac OS X operating system, released in 2001, included a new feature called 'the Dock' (see Figure 2.8). By default, the Dock shows only applications icons, not individual windows, and application names are only shown when the mouse cursor hovers over the icon. Users are also able to add *documents* to the Dock manually. Windows are added to the Dock only when they are minimised. Running applications are indicated by a small light (Mac OS X v10.5) or dot/triangle below the icons (earlier Mac OS X versions). When an application icon on the Dock is clicked all windows belonging to that application will be brought to the

foreground. The Dock has a so-called fish-eye effect; target sizes dynamically expand as the mouse cursor is closer to them, as can bee seen around the third icon in Figure 2.8.

Figure 2.8: The Mac OS Dock in Mac OS X v10.5 (Source: `http://support.apple.com`).

Command+Tab

Command+Tab on Macintosh operating systems is similar to Windows Alt+Tab, but the most important difference is that Command+Tab shows applications only, not all windows. When an application is selected using Command+Tab, all windows belonging to that application will be brought to the foreground.

Exposé

With the release of Mac OS X v10.3 a new window switching tool called Exposé (see Figure 2.9) was introduced. When activated, Exposé smoothly shrinks all windows so that they can be simultaneously viewed on the screen. Exposé provides different function keys for either showing all windows or just the windows for the current application (for example, when multiple PDF documents are opened). The spatial location of each window in the overview is influenced by its most recent location on the screen. While this relative positioning may assist users in visually seeking windows in the overview, the locations are not spatially stable between invocations if the locations of the windows change, and consequently the locations are unpredictable.

In 2010 Logitech introduced an *Application Switcher* functionality which works similar to Exposé.

Figure 2.9: Mac OS X's Exposé, showing all open windows on the screen simultaneously (Source: `http://support.apple.com`).

2.1.3 Empirical evaluations

Analysis of the efficiency and effectiveness of current commercial methods for switching between windows is often anecdotal. For example, some previous work deems the ordering of windows in Alt+Tab to be "very effective" (de Chiara et al., 2004, p. 366), but Alt+Tabbing is considered "tedious" by others (Grudin, 2001, p. 462). There are very few studies that evaluate the use, efficiency and/or effectiveness of the common commercially available window switching interfaces in a formal manner. Alt+Tab is found to be relatively fast when the number of windows is small (Kumar and Winograd, 2007), but performance decreases as the number of windows increases. Users make more errors when using the Windows Taskbar than when using Mac's Exposé, which may be due to the smaller target sizes on the Windows Taskbar (Kumar and Winograd, 2007).

Ramos et al. (2006) studied accessing occluded content in 2D drawings and introduced two novel techniques to access occluded objects. These two techniques, called 'Tumbler' and 'Splatter', are very similar to Windows Flip 3D and Mac's Exposé. Therefore, the results found by Ramos et al. (2006) are meaningful to examine in this context. Tumbler shows all objects in a 3D stacked view (like Windows Flip 3D), while Splatter (temporarily) separates the group of objects by 'splattering' them over the screen (like Mac's Exposé). An empirical study found a performance advantage of Splatter over the baseline control (a list with all objects by z-order), but no performance advantage for Tumbler.

2.2 Task-based approaches

Several studies indicate that people constantly multi-task and interleave various tasks during the work day (Bannon et al., 1983; Czerwinski et al., 2004; Dragunov et al., 2005; González and Mark, 2004). This is reflected in window use and management. For example, a 'writing a paper' task might contain a spreadsheet window, a statistical analysis package window, and a document into which the results are typed. Task switching occurs for several reasons (Bannon et al., 1983; Card and Henderson, 1987), such as users being reminded to do something else while performing a certain task, timesharing, practical limitations (like running out of file space), interruptions, and shifting to another project. Overall, task switching is less frequent than window switching (which takes place every 20.9 seconds on average, see Hutchings et al., 2004), but nevertheless a frequent activity: Czerwinski et al. (2004) found an average of 50 task shifts per week. The results in Czerwinski et al. (2004) inspired the development of GroupBar (Smith et al., 2003, reviewed later).

Many tools have been developed to support task management and switching by grouping windows by task. The underlying assumption is that users benefit from the close proximity (spatial or temporal) of grouped items.

The **Rooms** system (Henderson and Card, 1986; Card and Henderson, 1987) was the first switching interface that allowed users to manually partition space for different tasks by creating *virtual desktops*, or, *workspaces*. In Rooms, each 'room' is related to a different task, and the user can switch between tasks by going through (virtual) 'doors'.

Task Gallery (Robertson et al., 2000) is 3D task-based window switcher. Tasks appear as if they are artwork hung on the walls of a 3D virtual art gallery. In Task Gallery, a window can not appear in multiple tasks, something which is possible in the (similar) Rooms system.

GroupBar (Smith et al., 2003) is a modification of the Windows Taskbar that allows users to group windows into tasks. Clicking on one of these 'task buttons' causes all window belonging to that task to be restored and brought to the foreground. Task creation and management is manual and can be done by clicking and dragging items to or between groups. For example, users can drag a window button onto another button, which combines these two windows into a group,

and dragging a window button onto an existing group adds the window to the group. A user study reveals performance benefits as well as subjective preference of GroupBar over the (traditional) Windows Taskbar.

Activity-Based Computing (Bardram et al., 2006) also aims to support parallel work activities. Rather than working with a certain file or application, organisation is centred around the *activity* of the user, like tasks. Activity-Based Computing also offers a wide range of options to collaborate with other people. Activity-Based Computing allows the user to place any combination of windows into an activity, and these activities are placed on an 'activity bar' (similar to the Windows XP Taskbar) at the top of the screen. Ctrl+Tab switches between tasks (i.e., suspends one task and resumes the other), while Alt+Tab can still be used to switch between windows within one task. The authors evaluated Activity-Based Computing using a combination of methods, including think-aloud tasks, a questionnaire and an interview. The response from users was generally favourable, however, some fundamental issues were raised as well. Users struggled, in particular, with the lack of support for assigning the same window to more than one activity. The authors note that designing the system such that window can be classified under more than one task is problematic to achieve in "a simple manner".

With **Scalable Fabric** (Robertson et al., 2004) users can place individual windows, or groups of related windows, in the periphery of the screen. Windows placed in the periphery are made smaller to allow for enough space for the central focus region, in which the current window, or group of windows, is placed. Scalable Fabric is fully manual, and does not allow for a window to be placed in multiple tasks simultaneously. Nevertheless, it allows for users to be creative in their placement of windows in the periphery, so users can choose more 'fuzzy' arrangements (as done implicitly in Taskposé; Bernstein et al., 2008). Results from a user study suggest that users prefer Scalable Fabric over the Windows Taskbar, but details of this study are not described in Robertson et al. (2004).

Push-and-pull switching (Xu and Casiez, 2010) *implicitly* identifies tasks. Push-and-pull switching analyses window overlap and, based on the premise that users try to avoid or minimise the amount that windows belonging to the same tasks overlap, places window that do not overlap in the same group. An evaluation reveals that push-and-pull switching generates large performance benefits over other common window switching methods (direct click, the Windows Taskbar

and Alt+Tab). A draw-back of push-and-pull switching is that it is most likely less suitable for smaller displays, where most or all windows overlap (hence resulting in 'task groups' of one window each), or for users who tend to maximise all/most of their windows.

Taskposé (Bernstein et al., 2008) uses a fuzzy approach to defining tasks, using spatial proximity to illuminate task-based window relationships: windows that are frequently temporally adjacent drift closer to one another and those that are temporally distant drift apart. The size of the window representations reflects window importance, as determined by the WindowRank algorithm. Using the WindowRank algorithm a window becomes more *important* as other (important) windows switch to it often. The rationale behind Taskposé is that while task-based grouping of window in a switching interface can be beneficial, the creation of such groups is often problematic. First of all, it takes time, and second, some windows might be hard to classify, which Bernstein et al. (2008, p. 231) demonstrate with the following example: "*[...] consider a user who visits Amazon.com to purchase books for his child's birthday but gets distracted by a related item and starts browsing other items on the site instead. Should we still call this activity the* buying a birthday book *task? Should we instead put it in a catch-all* distracted *task?*" Therefore, rather than the "binary" approach that many task-based window switchers employ (either a window is in a task, or it is not; Bernstein et al., 2008), Taskposé's approach is "fuzzy". As a consequence, tasks are an emergent property in Taskposé, rather than explicitly constructed. The authors performed a week-long longitudinal evaluation of Taskposé. Participants reported that they found Taskposé most useful when the Window Taskbar had become very full as a result of many windows being open. None of the users ever reported Taskposé becoming too full or overcrowded. Finally, the authors report a lack of stability of the interface, as well as a lack of options for customisation.

With **WindowScape** (Tashman, 2006) the user can place window miniatures, which can be used for switching to windows, on the desktop, and these miniatures remain in spatially stable locations. Unfortunately, as these miniatures are on the desktop, they are often obscured by other windows. Therefore, WindowScape offers the possibility to temporarily bring the miniatures to the foreground without affecting the layout of the windows. WindowScape also *implicitly* supports task management by taking photograph-like snapshots of the window/screen layout

whenever the user expands one or more miniatures, or miniaturises one or more windows. These snapshots are placed on a time-line at the top of the screen and can later be returned to. The author acknowledges that this time-line is unstable.

SWISH (Oliver et al., 2006) is an algorithm to create a list of *related* windows based on semantic and temporal information. Semantic features are determined by looking for shared words in the window title, temporal proximity is determined by keeping track of the order in which windows are switched to. This semantic and temporal information is then used to build a list of related windows.

RelAltTab (Oliver et al., 2008) uses the SWISH algorithm to change the ordering of windows in Microsoft Windows Alt+Tab. In a user study, two RelAltTab prototypes were compared to the standard Alt+tab ordering as it is implemented in, among others, Windows XP. In one RelAltTab prototype the first three items are shown according to z-order, identical to the 'standard' Alt+Tab. Next, windows that are semantically related to the current foreground window are shown, followed by the ones that are temporally related, then followed by the rest of the windows according to z-order. In the second RelAltTab prototype the standard z-ordering is maintained, but colour indicates semantic and temporal relatedness of certain windows to the foreground window. Also, numbered key bindings allow the user to switch to semantically related windows more quickly. A user study revealed that the second prototype leads to faster window switching times than the standard Alt+Tab for some tasks, but prototype one was slower than the other two methods for some other tasks. Also, prototype two led to higher user satisfaction ratings than the others.

In general, there are several problems associated with task-based grouping of windows, first of all related to whether the interface requires manual management of tasks or provides automatic task identification and management. *Manual* classification of windows into tasks (as implemented in GroupBar, Activity-Based Computing, Scalable Fabric and Task Gallery) means that users must carry out explicit actions to gain potential benefits and it creates additional cognitive overhead for the user (Kaptelinin, 2003; Dragunov et al., 2005). *Automatic* grouping of windows into tasks, however, is risky, because it might not match with the users' classification. The primary limitations of such automatically adaptive systems are that they can incorrectly predict the user's intention and that users can fail to understand or anticipate the system's adaptation (Shneiderman, 1997). When this

happens users must resort to a time-consuming visual search of candidate targets.

Another design issue for window switchers that use a task-based grouping is the fact that a window might be used in several tasks, i.e., some windows are likely not to be 'task-specific' at all. For example, generic applications, such as a web browser or an e-mail client are likely to be used across tasks, rather than in one specific task. If a task-based window switcher only allows for exclusive grouping (i.e., a window can only be associated with one task, e.g., GroupBar, Scalable Fabric, Task Gallery) the user is sometimes forced to make 'impossible' choices, or open multiple windows for these applications, which some users find unnatural or difficult: "*[still] trying to get used to having multiple internet windows open*" (Smith et al., 2003, p. 40). Conversely, some windows might not be associated with a specific task at all, but some task-based window switching interfaces nevertheless 'enforce' a classification. This is reported in Bardram et al. (2006, p. 219): "*[the] worst thing? Well [...] if you have to put everything into activities, then you need to constantly consider 'where does this one belong'. In many situations something just appears quickly and then you start up some application and do some things in it. [...]*".

2.3 Window organisation

In most common operating systems windows will usually be (partially) overlapping each other (Myers, 1988). This overlap means that windows low in the *z*-order are often (completely) obscured by other windows, and can therefore not be reached directly (by clicking on the relevant window using the mouse). Also, it implies that the user has to continuously switch back and forth between windows when the task at hand requires information from, or interaction with two or more windows (Baudisch and Gutwin, 2004).

In window management research two distinct solutions to this problem can be identified: (1) leaving the window positions 'as is', but assisting users in accessing and revealing occluded content and (2) rearranging windows such that multiple windows are visible simultaneously, e.g., by tiling all windows. This dichotomy between overlapping and tiling approaches was already observed in the 1980's (Bly and Rosenberg, 1986; Myers, 1988). The following sections review the research related to these two approaches.

2.3.1 Support for interacting with overlapping windows

Researchers who investigate techniques for interacting with overlapping windows often do this in the context of copying or moving objects between windows (e.g., Dragicevic, 2004; Chapuis and Roussel, 2007; Faure et al., 2009), as this is a typical example where the user quickly and briefly needs to switch between windows. Because of the nature of this interaction, Dragicevic (2004) argues that (1) rearranging windows such that they are next to each other requires too much effort, and (2) switching back and forth between the windows by means of a window switching interface is too cumbersome.

Many proposed techniques for dealing with overlapping windows draw on analogies with paper, such as rotating and peeling/rolling back occluding windows (Beaudouin-Lafon, 2001; Chapuis and Roussel, 2007; Dragicevic, 2004).

Chapuis and Roussel (2007) observe that user interest is not by definition the window that is clicked in. For example, a window is sometimes only used to (quickly) copy something, after which the user wants to return to the window of primary interest, where the item/text is pasted. The techniques proposed by Chapuis and Roussel (2007) detect which window is of interest based on the location of the mouse cursor, and either (temporarily) roll back occluding windows or change the stacking order of the windows to bring that particular window to the top of the z-order. The rolling back technique in Chapuis and Roussel (2007) is similar to the work by Dragicevic (2004), who reports on a technique where an overlapping window is partially folded away to reveal underlying objects. The study by Faure et al. (2009) proposes, among others, a technique to control the stacking order of layers of non-overlapping windows, similar to the re-stack technique by Chapuis and Roussel (2007) and the window switching technique in Xu and Casiez (2010).

A different solution to address the problems with overlapping windows is to make the top window(s) semitransparent. However, as the traditional technique for achieving such transparency (*alpha blending*) can render window content poorly legible, Baudisch and Gutwin (2004) propose a technique called 'multi-blending', which is aimed at preserving the most relevant features of both foreground and background content.

2.3.2 *Support for displaying multiple windows*

Several novel window management tools provide support for displaying multiple windows on the screen simultaneously. When such a layout is used a direct click in a window is possible relatively often, thus eliminating the need for a external widget like a window switching interface.

Elastic Windows (Kandogan and Shneiderman, 1996, 1997) is a system that aims to tile all open windows on the screen in a hierarchical space-filling manner. Kandogan and Shneiderman (1996) argue that overlapping window techniques causes too much (cognitive) load for the user, and that users could benefit from having all windows visible on the screen simultaneously. Also, they argue that their technique provides a close analogy with an actual desk layout, as Malone (1983) found that people like to organise their desk spatially. With the Elastic Windows system windows automatically accommodate for changes in the size of other windows: increasing the size of a window automatically causes the other windows to shrink, while utilising all screen estate.

Tiling windows is not possible on smaller screens (Myers, 1988), as sufficient screen estate is required to display all windows simultaneously. Therefore, techniques have been developed to allocate sufficient space to the focal window, while shrinking, moving, or only partially showing other windows. Hutchings and Stasko (2002) propose techniques (expanding and shoving windows) that allocate a large amount of screen space to the current focal window, while preserving (but shrinking and moving) the visible portions of all other windows. Window snipping (Hutchings and Stasko, 2004b) shrinks windows by 'snipping' them such that only the relevant region of the window (as defined by the user) is visible. Wincuts (Tan et al., 2004) also allows for partial content of multiple windows to be visible on the screen simultaneously, similar to window snipping (Hutchings and Stasko, 2004b), except that the 'cuts' by Tan et al. (2004) can not be interacted with, while the 'snips' by Hutchings and Stasko (2004b) are interactive.

Another solution to display multiple windows at the same time is to not necessarily show *all* windows, but only the *relevant* windows for the task at hand. Haraty et al. (2009) propose AdWiL (Adaptive Windows Layout Manager), which automatically detects which task the user is working on and generates appropriate window layouts for the windows involved in this task (AdWiL automatically de-

termines window importance and relatedness). Once the system has determined several appropriate layouts, it displays a small preview of the candidate layouts in the corner of the screen and the user can select to apply a layout.

2.3.3 Comparing overlapping and tiled approaches

The results found by Chapuis and Roussel (2007) show an efficiency advantage of rolling back occluding windows and automatic restacking of windows over layouts where the windows are either overlapping (without any additional techniques) or non-overlapping (such as tiled views). Also, the results show an efficiency advantage of non-overlapping over overlapping (without any additional techniques) layouts. This latter result is similar to the finding from the empirical study by Kandogan and Shneiderman (1997), which reveals performance benefits of the (tiled) Elastic Windows system over a layout with overlapping windows. Also, Hutchings and Stasko (2007) provide an evaluation of 'snipping' window behaviour in multi-monitor environments and find space and time efficiency gains compared to an overlapped approach.

Finally, other work suggests that the suitability of certain window management techniques depends on the task at hand. Bly and Rosenberg (1986) found that tiled layouts are more suitable (in terms of faster performance) for tasks that require little window manipulation (such as scrolling), while overlapping windows are more suitable for tasks that require a lot of window manipulation (e.g., when the window content does not automatically adapt to the window's shape and size).

2.4 Eye-gaze input

Some window switching research is focused on the development of systems that respond to eye-gaze, like EyeWindows (Fono and Vertegaal, 2005) and EyeExposé (Kumar et al., 2007) which is based on Exposé.

With the EyeWindows (Fono and Vertegaal, 2005) system all windows are displayed simultaneously, similar to the Elastic Windows system by Kandogan and Shneiderman (1996). The user selects a window by looking at it and pressing an activation key (e.g., the space bar), after which the window is dynamically enlarged, and the surrounding windows accommodate this change by shrinking

proportionally. Fono and Vertegaal (2005) demonstrate that their EyeWindows system outperforms an Exposé-like system (that used mouse input).

EyeExposé (Kumar et al., 2007) functions like Mac OS X's Exposé, except that it responds to eye-gaze rather than requiring mouse input. In an evaluation EyeExposé was found to be slower than Alt+Tab when four windows were open, but faster when 12 were open. The evaluation found no significant difference between EyeExposé and Exposé.

The different results of the evaluations of EyeWindows and EyeExposé (EyeWindows outperforms Exposé, while EyeExposé does not, even though EyeWindows and EyeExposé are quite similar systems) is probably due to subtle differences in experimental procedure: in the EyeExposé study participants were able to keep one hand on the mouse, but they were unable to do so in the EyeWindows study, as typing was required between the window switches. This means that the response times for the Exposé condition in the EyeWindows study was increased by the time it took to move a hand from the keyboard to the mouse.

Chapter III

Related Work

This chapter reviews various work related to window switching. First, an overview of previous work on how people interact with windows is provided. This includes window switching, display space management and the use of large displays and multi-monitor setups. Next, this chapter presents an overview of work related to visual search, spatial cognition, and motor skills.

3.1 Studies of window use

This section examines research related to (1) window switching (for details about particular tools people can use to interact with windows including window switching tools, see Chapter 2), (2) display management (e.g., how people organise windows on the screen), and (3) large displays and multi-monitor setups.

3.1.1 Switching between windows

Switching between windows is a very common task. A study by Gaylin (1986) showed that window switching activities are far more frequent than window creation, deletion, or geometry management. Hutchings et al. (2004) found that the average time any window is active is a mere 20.9 seconds and that the median activation time is 3.77 seconds.

Studies that investigate how people switch between windows are rare. Hutchings et al. (2004) found a relationship between the method used for switching between windows and the number of monitors a user has; users that have multiple monitors are more likely to use a direct window action (such as a click on the target window or minimising the top window) and less likely to use the Windows Taskbar to switch between windows. The authors suggest that multiple monitor users use the Windows Taskbar less often to switch between windows because the user has to traverse a large distance to reach it.

A longitudinal log study by Mackinlay and Royer (2004) identifies 'window thrashing' behaviour: "*the rapid manipulation of windows caused by limited display resource*" (p. 1). The authors identify a "time-multiplexing" strategy, where the user rapidly switches between windows that cover 100% of the monitor. Conversely, "space-multiplexing" is a strategy where the user frequently changes the size of windows, for example, to reveal content of an underlying window. Time-multiplexing is more common than space-multiplexing, which is most probably due to the fact that space-multiplexing requires windows to be resized, which the authors note is an 'expensive' task. Finally, the authors note that 'window thrashing' is probably less common on multi-monitor setups, as window thrashing is indicative of the display being (too) small.

3.1.2 Display space management

In this section, research related to how many windows users have open and visible, how these windows are arranged on the screen, and how much space is left 'empty' is reviewed.

In terms of how many windows users have open, Smith et al. (2003) report that single monitors users on average have 4 windows open at once, dual monitors users 12 windows and triple monitor users 18 windows. However, a window that is *open* is not necessarily *visible*. For example, it can be in a minimised state or be obscured by other windows. Hutchings et al. (2004) logged the windowing activities of 39 people for a three-week period. They report that single monitor users on average have 3.5 windows visible on the screen, users of small multiple monitor setups (less than 3 million pixels of screen estate) 4.1 windows, and users of large multiple monitor setups (more than 3 million pixels of screen estate) 6.8 windows. Also, 78.1% of the time users had eight or more windows open.

Hutchings and Stasko (2004a) interviewed twenty computers users about their management of display space. The authors identify three different window management styles: *maximisers*, who maximise most of their windows, *near maximisers*, who manually resize windows to almost full size, while leaving some space unoccupied for either the desktop or smaller windows such as instant messaging systems, and *careful coordinators*, who have many windows visible simultaneously and hardly ever maximise windows.

Chapter 2 discussed the dichotomy between overlapping and tiled approaches to window management. Hutchings and Stasko (2004a) report that people hardly ever tile their windows. This finding, that people usually employ a overlapping window management technique (rather than tiling), stresses the importance of adequate window switching tools.

Hutchings et al. (2004) report a correlation between the total amount of screen *space* and the amount of *time* with empty space: if there is more screen estate available, people more often leave some of this space uncovered. However, the authors found no correlation between display type (single monitor, small multiple monitor setup, or large multiple monitor setup) and the *amount* of empty space.

Finally, Hutchings and Stasko (2004a) identify several other window management behaviours, such as intentional hiding of windows to reduce distractions and for privacy-related reasons, and a relationship between input device and management of screen space (e.g., when there is limited space for the mouse to move).

3.1.3 Large displays and multi-monitor setups

Computer users use increasingly larger monitors, and multi-monitor setups are becoming increasingly more common (Bi and Balakrishnan, 2009). Also, the observation that common monitor setups cover only 10% of the visual field of users (Grudin, 2001) has inspired some researchers to investigate the usefulness of large and very large displays (Bi and Balakrishnan, 2009). Therefore, this section examines the research related to large displays and multi-monitor setups.

Grudin (2001) investigated how people use multi-monitor setups by means of a field study, including observations of the work space and interviews, of 18 users. His findings indicate that on dual monitor setups the secondary monitor is used for secondary tasks, and for peripheral awareness. In other words, each monitor serves a specific function, rather than all screen estate being treated as if it were one large monitor. This is related to the observation that dual monitor setups do not lend themselves well to spanning a window across two monitors, as the bezels interrupt the displays space. Indeed, Grudin (2001) reports that users rarely straddle a window across both monitors. However, this separation between monitors is not necessarily bad, because, as the author points out, it facilitates partitioning of the workspace. Finally, participants in Grudin (2001, p. 462)

indicate that a dual monitor setup helps them to "*[escape] from the need to Alt-Tab*", as multiple windows can be viewed simultaneously.

Results similar to the study by Grudin (2001) were found by Bi and Balakrishnan (2009). They report that dual monitor users often do not distribute their activities equally across both monitors, with 71% of mouse events taking place on one monitor and 29% on the other. One monitor is used as the focal region and the other monitor as the peripheral region; the focal region is used for primary tasks (e.g., document writing), where users spend the most of their time on, and the peripheral region is used for secondary tasks. A typical example of such a secondary or peripheral window is the email inbox. Hutchings et al. (2004) found that the window containing the e-mail application is (1) invisible less often, (2) *fully* visible more often, and (3) *active* when fully visible less often for multi-monitor users compared to single monitor users.

Large displays and multi-monitor setups have several advantages for the user: (1) large displays have been shown to increase awareness of secondary tasks and assist multitasking (Bi and Balakrishnan, 2009; Czerwinski et al., 2006) and (2) users perform tasks quicker on large and multi-monitor displays compared to single-monitor setups (Czerwinski et al., 2003; Kang and Stasko, 2008).

However, there are disadvantages to large displays and multi-monitor setups. For example, windows can pop up in unexpected places, and placement of windows requires more attention to ensure they do not cross bezels (Bi and Balakrishnan, 2009; Czerwinski et al., 2006). Some dual monitor users report that they find the head movement required to shift attention from the primary to the secondary monitor uncomfortable, and distal access to windows and icons can be problematic on large displays (Bi and Balakrishnan, 2009).

Finally, Hoffmann et al. (2008) studied window switching on large screen setups, and locating the window receiving focus in particular. For example, switching to a window using Alt+Tab will bring that window to the foreground, but its x,y-coordinates do not change. Therefore, the user can have trouble locating the window that received focus, as it is sometimes outside the region where the user is focusing. Hoffmann et al. (2008) demonstrate that highlighting the target and presenting graphical trails to the target alleviate this problem.

3.2 Visual search

When looking for a specific button on the Windows Taskbar or a window in Mac's Exposé the user has to *search* for it. Put formally, visual search is the process of finding a target among a field of distractors (or non-targets).

If none of the items in a layout stands out, the average time to find an item in a set of N items is $\frac{N}{2}$, because on average the target will have been found after half the items have been inspected (Neisser et al., 1964). For example, this has been found to be the case when people search in a phone book or computer menus (Lee and McGregor, 1985). If the items are arranged evenly and coherently visual search tends to be top to bottom and left to right. For less structured views the search tends to be much more random (Wickens, 1992). Also, research using eye-tracking devices reveals that people have little memory for what items they have already inspected during a visual search, often returning to areas that have previously been searched (Snowden et al., 2006).

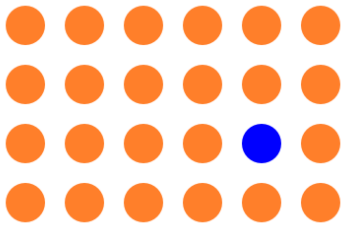
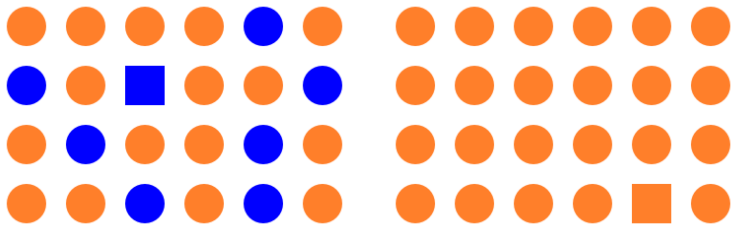


Figure 3.1: Visual pop-out: the blue dot stands out between the orange dots.

However, visual search is not purely a top-down process, where a person has the target in mind and then sets off to (linearly) search for it (Wickens et al., 1998). Bottom-up processes, where the visual stimulus influences the search process, take place as well. For example, this is the case if one of the items 'pops out'. Certain features of an item can make it stand out between other (non-target) items (for example, see Figure 3.1). This 'pop-out' effect can dramatically reduce search time. Colour, size and brightness are among the many factors that can make an item pop out (Treisman, 1986).

Feature integration theory (Treisman and Gelade, 1980) can explain the effect referred to as 'visual pop-out'. Feature integration theory proposes that there are two stages in visual search, a pre-attentive *parallel* stage (called the feature search)

followed by a more selective *serial* stage (called the conjunction search). Pop-out is believed to take place in the pre-attentive stage. For example, when looking for a blue item between orange distractors, this proves to be a very easy task: the blue item 'pops out' visually. The response time does not depend upon how many distractors there are, hence this is called a parallel search, where many items are analysed at the same time. For more complex searches, like when multiple features (e.g., colour, size, orientation) are combined, the search is believed to be serial, where paying attention to each element in turn is required. In this case, the response time increases when the number of distractors is increased.



(a) (b)

Figure 3.2: Guided search: it is much easier to find the blue square in (a) than it is to find the square in (b).

Some regard the distinction between serial and parallel search in feature integration theory as too simplistic (Snowden et al., 2006), as explained here:

> "*It is a curious feature of these models that the parallel processes seem to have very little influence on the subsequent serial processes. In the standard feature integration model, the parallel processes can identify targets on the basis of a single feature. However, if they do not find a target, none of the information that they have gathered is used by the serial processes, even if that information might be useful.*" (Wolfe et al., 1989, pp. 419-420)

Wolfe et al. (1989) therefore propose a *guided search model*, which is an extension of feature integration theory, but elaborate this model with the notion that during the pre-attentive stage of analysing a scene, selecting appropriate objects for the second stage of analysis (the more advanced processing of the scene) is already guided. For example, attention can already be guided to an appropriate subset of

all items during the pre-attentive stage, decreasing the overall response time. For example, see Figure 3.2: it is much easier to find the blue square in see Figure 3.2a than it to find the square in see Figure 3.2b, as colour is a guiding attribute. Wolfe and Horowitz (2004) provide an overview of which attributes are believed to guide attention.

Figure 3.3: Two icons that look very similar, both in terms of shape and colour. Finding either one of them (in a set of icons) is hard when the other is present as a non-target, because of the high target/non-target similarity.

In both feature integration theory and guided search, dissimilarity of the target to the non-targets plays an important role. Not surprisingly, it has been found that finding a target becomes harder when the target/non-target similarity increases (also see Figure 3.3) or the non-target/non-target similarity decreases (Duncan and Humphreys, 1989). Indeed, research has shown that icons sets with icons distinguishable on a few features help users in their search (Byrne, 1993). Even more, icons sets with icons that largely contain all the same features or the same combinations of features, are found to perform worse than blank icons (Byrne, 1993) (text descriptors of the items were also available in this study). However, with the nearly infinite number of computer icons available nowadays it seems impossible that none share (a lot of) features with another icon. Therefore, Lewis et al. (2004) propose a system that automatically generates distinctive icons. Meanwhile, the development of commercial window switching interfaces has moved toward more rich visual representations of windows (see Chapter 2).

In all, if a window switching interface relies on users performing a (linear) visual search for the target, efficiency can be expected to be low. However, if a user knows or has a correct expectancy of where the target will be, this reduces or even eliminates the need for visual search (Parasuraman, 1986). One way to achieve this is by keeping items in spatially stable locations, so the user has the opportunity to learn item locations. The importance and the effects of spatial stability are reviewed in the next section.

3.3 Spatial cognition

Spatial cognition is the overlapping term for learning, knowing, revising and remembering information about our spatial environment. We know the route to work, the place where we put our keys and where to find the coffee in the supermarket. However, changes in our environment can disrupt this spatial knowledge: a renovation of the supermarket will lead to a painstaking search along the aisles until you find your favourite brand of coffee again. This not only applies to the built environment around us, but also the desktop environment on your computer. You know where the icon for launching your web browser is, and any disruptions to this can be frustrating and will slow you down as you now need to search for the icon (also see previous section).

Figure 3.4: Desk organisation.

The closest analogy in the physical environment to the desktop computer metaphor is workspace and desk organisation. Malone (1983) conducted a study of how people organise their desks (see Figure 3.4). Malone (1983) found that the spatial location of 'piles' of documents is particularly relevant for the user. Several studies (e.g., Barreau and Nardi, 1995) found that people prefer location-based finding on the computer desktop as well. However, despite parallels between the desktop environment on the computer and an actual desktop, there are also large differences in terms of how they are and can be used. A stack (or pile) of windows can be viewed as being similar to a stack of books and papers on your desk, yet

it does not afford the same interactions. While on the actual desktop an item can easily be pulled from the middle of the pile, and the user can quickly 'peek' at an item in the pile by lifting the items on top of it, the (standard) GUI desktop does not afford these types of behaviours. Some interfaces have been developed to make the GUI desktop behave more like a 'real' desktop (e.g., Agarawala and Balakrishnan, 2006).

For window switching interfaces, providing a layout that is spatially stable could greatly benefit user performance. The positive effects of spatial constancy on user performance have been recorded since the first interface design guidelines. Hansen (1971) stated that operations can be optimised by supporting display inertia (spatial stability in item placement) because doing so facilitates predictability and supports the use of muscle memory. Many subsequent studies have demonstrated superior performance of spatially stable interfaces over a wide variety of alternatives. For example, Teitelbaum and Granda (1983) examined the impact of fixed versus unstable positions of various widgets on a computer screen, such as a text entry regions. Slower reaction times were found for the non-constant user interface. In general, fixed stable item placement allows users to decide about item locations, with performance characteristics that increase logarithmically with the number of items (Cockburn et al., 2007). This is in contrast to searching for items in unstable displays, the time of which linearly increases with the number of items.

Several studies have found that users are able to learn the locations of a large number of items in a user interface when the locations do not change. For example, Cockburn et al. (2007) used a 'frost-brushing' interface, which successfully 'forced' users to learn item locations. This is similar to the findings by Ehret (2002), who found that when items had less representative labels, users learnt their locations better. Another example can be found in a series of studies using an interface for organising web pages called Data Mountain (Robertson et al., 1998; Czerwinski et al., 1999). In the first study, participants were requested to place 100 snapshots of web pages in a desktop 3D environment and perform some retrieval tasks. After a few months, in which participants had not seen their layout, they were again given several retrieval tasks in the same layout they created months before. The retrieval times were not significantly slower than in the initial study. Users were also presented with a view where the thumbnail images were replaced with blank icons. After an initial drop in reaction times, users were soon

equally fast in retrieving the web pages in the 'thumbnail-off' view. This result indicates that people can learn the location of large numbers of items.

Finally, some studies have demonstrated a statistical interaction between the representativeness of the icon or label and reliance on information about the location of the icon (Moyes, 1994; Ehret, 2002): the less representative, or the more abstract, an icon or label is, the more users rely on knowledge about the location of the icon. This suggests that participants do not remember location well when they can rely on recognition instead (Moyes, 1994). Ehret (2002), in contrast, demonstrated that users learn locations regardless of representation, but that the speed and accuracy of spatial learning is aided by abstract representations which 'force' users to attend closely to spatial location in the absence of other cues.

3.4 Motor skills

Many window switching interfaces allow for mouse-input by clicking on the target item. The time it takes the user to point to an item is mathematically captured by Fitts' Law (1954). Fitts' Law is a well-known and thoroughly tested rule of target acquisition time. Fitts' Law states that the time required to move to a target (movement time, *MT*) is a function of both the width of the target (*W*) and distance to the target (amplitude, *A*). More precisely, the following relationship has been proposed and validated (MacKenzie and Buxton, 1992):

$$MT = a + b \times log_2(\frac{A}{W} + 1) \tag{3.1}$$

The logarithmic term in Equation 3.1 is called the 'index of difficulty' (*ID*) and it is expressed in 'bits'. There are several alternative formulations for Fitts' Law, where the *ID* is calculated differently. For example, the original Fitts' Law paper (Fitts, 1954) uses $2 \times A/W$ rather than $A/W + 1$ (MacKenzie and Buxton, 1992). MacKenzie and Buxton (1992) show that the formulation shown in Equation 3.1 has the advantage that the index of difficulty is always positive. Negative *ID* values result in negative movement time predictions, which is clearly undesirable. The fact that several different variations of Fitts' Law are used for HCI research is viewed as problematic by some researchers (Drewes, 2010).

In Equation 3.1, a and b are empirically derived constants, and vary with the type of pointing device (e.g., mouse, track-ball), the control-display gain (the ratio of movement of the device to movement of the proxy on the screen), and individual user differences. The values for a and b are typically empirically derived by presenting users with a pointing task for different values of A and W. When fitting a straight line through the observed values for movement time, a and b are the intercept and the slope of the line, respectively. Once the values for a and b have been derived, Equation 3.1 can be used to predict movement times.

The original application of Fitts' Law was only for one-dimensional targets. However, targets on the computer screen will usually be two-dimensional, rendering the target width (W) ambiguous. MacKenzie and Buxton (1992) show that a good interpretation of target width (W) of two-dimensional rectangular targets is to take the smallest of the height and width of the target.

Some research has focused on the Fitts' Law acquisition time of dynamically expanding targets (McGuffin and Balakrishnan, 2002), demonstrating that even target expansion that only takes place when 90% of the distance to the target has already been travelled still leads to improved acquisition times. Interestingly, other research has shown that this beneficial effect even exists by just making the target *look* bigger while not increasing its actually (clickable) area (Cockburn and Brock, 2006).

When acquiring a target at the edge or in the corner of the screen the user can (intentionally) 'overshoot' the target, resulting in movement times lower than predicted by Fitts' Law (Farris et al., 2001).

Setting aside specifics about the values of a and b and the most appropriate way of determining ID, the general 'gist' of Fitts' Law is as follows: the larger and/or the closer the target, the quicker it can be acquired.

Chapter IV

PyLogger and Window Watcher: Tools for Studying Window Use

Having a good understanding of window use is critical for design refinement, yet there are few recent studies of what users actually do. Studies of window use are difficult for two reasons. First, gathering data about window use can be difficult, as automated log based analysis is preferential, but developing such software can be complex. Second, recording data at the keystroke or system level creates very large log files, which are difficult to analyse and interpret (Ivory and Hearst, 2001; Beauvisage, 2009; Hilbert and Redmiles, 1999).

This chapter describes the logging tool *PyLogger* and the visualisation tool *Window Watcher*, with which data regarding window use, such as the number of windows, how windows are arranged on the screen and how users switch between windows can be gathered and analysed. This approach, which combines quantitative statistics and visualisations, is supported by Perer and Shneiderman (2008).

PyLogger is a client-side logging tool and is currently designed for the Microsoft Windows XP and Vista operating systems. *PyLogger* requires no user input during use, nor any modifications to the users' system. *Window Watcher* is a data visualisation tool that helps to interpret the low level event logs generated by *PyLogger*. *Window Watcher* affords a variety of valuable insights into the spatial and temporal aspects of window use. The tool can, for example, 'replay' all window actions.

This chapter first describes the architecture and implementation of *PyLogger*, including some examples of how to use Python to extract window events. Next, the features of *Window Watcher* are described, including examples of the ways that it summarises and elucidates window use.

The results of the longitudinal study that was conducted using *PyLogger* are presented in Chapter 5.

4.1 Contributions and findings

- a description of the architecture and implementation of the logging tool *PyLogger*;

- examples of how to use Python to expose the Windows API and GUI, and to capture the low level keyboard and mouse events in Windows, which should be useful for researchers wishing to replicate or update this study;

- features of the visualisation tool *Window Watcher*, and examples of how a visualisation tool like *Window Watcher* can expose aspects of window use, which should inspire other researchers with similar data sets.

4.2 Data collection methods

Methods to collect data about people's behaviour include automated logging of the behaviour, observing the behaviour, or asking people about their behaviour (so-called self-reports) by means of a questionnaire or interview. For the purpose of studying window use, however, automated logging of the behaviour is preferential for several reasons.

First, automated logging of behaviour can be easily deployed over longer periods of time. Such longitudinal analysis is often desirable because individuals patterns of behaviour change as they move between tasks and external pressure levels, meaning that a short 'snapshot' of interaction is likely to misrepresent real behaviour. Also, in general, individual or short snapshots are not very informative for exploring window use, as these behaviours are often only interesting when they are viewed in context. For example, the *individual* act of switching to a window is not informative, but the finding that a user does this every x seconds is. Longitudinal analysis allows for findings to be explored in the context of what happened before and next (Pettigrew, 1990). Longitudinal analysis can also catch rare events which might have been missed in shorter studies.

Second, automated recording of the behaviour facilitates large sample sizes, something which direct observation or interviews, for example, do not facilitate. These large sample sizes are necessary because users differ widely in their work practices (e.g., tidy versus chaotic desktops, both real and virtual). Users also

differ widely in the display technologies used (e.g., large multi-monitor environments versus small laptop screens), again necessitating large sample sizes with a variety of different display types.

Third, automated logging can provide precise statistics, including exact timings (Holzinger, 2005). Previous studies have already found that window switching is very common, with the median time a window is active being only several seconds (Hutchings et al., 2004). Automated logging can accurately keep track of these frequent events.

Fourth, log-based analysis is non-obtrusive, which is important as any disturbance could lead to false results (Holzinger, 2005), e.g., the user behaving in a different way than he/she normally does.

Though automated logging is preferential for the current type of study, a downside is that little to nothing can be extracted about the *context* of observed behaviour, or the (subjective) user experience (Alexander et al., 2008).

Two other common data collection methods, observational studies and self-reports, are less suitable for the current type of research. Observational methods, though able to provide information about the context of behaviour, are impractical for large-scale and long-term deployment. Also, some properties of windows on the screen, such as the relative position of obscured windows, can only be extracted by querying the system, not by observation. Furthermore, observational methods are obtrusive, and it is easy for the observer to lose concentration or miss events (Preece et al., 2007). Self-reports, in particular questionnaires, could be distributed to a large user population and are not obtrusive. However, such self-reports of behaviour are potentially unreliable. Unless behaviours are extremely rare or of high importance, it is unlikely for people to have detailed and/or accurate memories of them (Schwarz, 1999; Tourangeau, 2001), and memory for computing events is no exception to this (Czerwinski and Horvitz, 2002). Also, people are often not even aware of the way they perform mundane and day-to-day activities (such as window use), as they run off without much conscious awareness (Roediger, 1990).

4.3 Visualisation tools

Powerful tools are required to explore and analyse the often voluminous log files generated by automated logging (Preece et al., 2007). One key strategy to assist the analysis of large data sets is to use visualisations (Card et al., 1999). Visualisations tools can help to gain insights into behaviour, observe certain behavioural patterns (Keim, 2002; Shneiderman, 2002) and reveal data patterns that would otherwise have been impossible or difficult to find (Card et al., 1999; Gray et al., 1996). Also, visualisations can inspire the analysis, leading to "*answering questions you didn't know you had*" (Plaisant, 2004, p. 111).

In the specific context of how people interact with computers, visualisations have proven very useful in studying how people navigate in and between websites (Chi, 2002; Cugini and Scholtz, 1999; Eick, 2001; Hong and Landay, 2001). Also, visualisations of interactions with windows and/or the screen have proven to be intuitive and powerful, as can be seen in the visual depiction of mouse events in Bi and Balakrishnan (2009) and Atterer et al. (2006).

4.4 PyLogger

This section describes *PyLogger*, a client-side logging tool for Microsoft Windows XP and Vista that records information about window use in an unobtrusive manner. To effectively capture all aspects of window use, a logging tool designed for this purpose needs to be able to do the following:

- record information about all windows, including their position and state (e.g., minimised);
- recognise changes in the focal window: (a) a new window becoming the focal window or (b) the focal window changing size/position;
- record low-level user actions such as key presses and mouse button clicks.

PyLogger records when a new window gets focus and what action caused it (e.g., a click on a Taskbar button, or launching an application using a Quick Launch button). It also records when the focal window is moved or resized. Every time a change is recorded *PyLogger* logs all windows' position and state.

PyLogger is written in Python 2.6, and it uses the *Python for Windows* package[1] to expose the Windows API and GUI to Python and the *PyHook* package[2] to wrap low-level mouse and keyboard hooks in the Windows Hooking API.

To be able to use *PyLogger*, users do not have to make any changes to the computer other than to install *PyLogger* itself; *PyLogger* is launched at system start-up and placed in the system tray. *PyLogger* has not been found to conflict with any other software, including firewalls and spyware/virus scanners. No participants in the study reported an impact of *PyLogger* on their computer's performance.

Log files are saved in text format (txt) and are therefore viewable and editable in most common document editors. To minimise disk space and to prevent data loss if the output file is corrupted, the output text file is compressed (zip) at the end of each day and a new output file opened.

The following sections describe how *PyLogger* accesses window information, detects change in the focal window and records user actions.

4.4.1 Accessing window information

Figure 4.1 shows how to expose the full list of currently open windows, including the window handle (a unique numerical identifier), the window title, the window rectangle (the *x*,*y*-coordinates of the top-left and bottom-right corners of the window), the window owner and the window class. Most windows are not *owned* by another window; a typical example of a window that *is* usually owned by another window is a dialog box (such as the 'Save as...' dialog box). Window class helps to determine the application, but not all window class names are straightforward (e.g., Microsoft Word 2003 windows have class name 'OpusApp').

Without the *IsWindowVisible* check in the *windowEnumerationHandler* method many redundant windows are included in the list. These are (system) windows that are never visible for the user, and therefore not relevant for the current study.

The enumerate function *EnumWindows* returns all windows ordered by their position in the *z*-order, from top to bottom, which is also used in the Alt+Tab window (see section 2.1.1). Finally, minimised windows have the negative *x*,*y*-coordinates -32000, -32000.

---

[1] `http://sourceforge.net/projects/pywin32/`

[2] `http://sourceforge.net/projects/pyhook/`

```
from win32gui import IsWindowVisible, GetWindowText,
    GetWindowRect, EnumWindows

def windowEnumerationHandler(hwnd, windowList):
    if IsWindowVisible(hwnd) > 0:
        title = GetWindowText(hwnd)
        rectangle = GetWindowRect(hwnd)
        owner = GetWindow(hwnd,4)
        clss = GetClassName(hwnd)
        windowList.append((hwnd, title, rectangle, owner, clss))

def getWindowList():
    windowList = []
    EnumWindows(windowEnumerationHandler, windowList)
    return windowList
```

Figure 4.1: Using the win32gui module to enumerate all windows.

4.4.2 Detecting change

PyLogger uses a polling loop to check whether the focal window has changed every 100 milliseconds. A change in the focal window is defined as another window becoming the focal window, or the size/position of the focal window changing. Using the threading module this is very easy to implement, see Figure 4.2.

```
from threading import Timer
def pollingLoop():
    checkTopWindow()
    t = Timer(0.1, pollingLoop)
    t.start()
```

Figure 4.2: Using the threading module to create a polling loop.

Whenever the polling loop detects a change the output method is called, which appends the following information to the output text file:

- The date and time of the event;
- What happened that triggered the logging tool to respond (either a new focal window was detected or the focal window changed size and/or position);
- The *lastAction* value (see next section);

- How many windows are open (including minimised windows);

- Information about the focal window: window handle, window title, window position, window owner and window class;

- A list of all windows in *z*-order, with the following information about the windows: window handle, window title, window position, window owner and window class.

4.4.3 Recording user actions

PyLogger continuously keeps track of mouse button clicks and keystrokes and stores the last user action it has 'seen' in the *lastAction* variable (i.e., the *lastAction* variable is constantly overwritten at each user action). Next, if a window focus change or window move/resize is detected this *lastAction* value will *usually* be the action that triggered the change (some inaccuracy is inevitably introduced as the polling loop only checks for change every 100 milliseconds).

PyHook is used to hook into these low-level actions (for an example of how to hook left mouse button clicks, see Figure 4.3). Some examples of the values the *lastAction* variable can have are 'keyboard ['Alt', 'F4']' (the key combination Alt+F4) and 'left [216, 768]' (the left mouse button was clicked at screen coordinates (216, 768)). When the user clicks on a Taskbar button *event.WindowName* is 'Running Applications' (in the English version of Microsoft Windows).

```
1  import pythoncom, pyHook
2
3  def OnMouseLeftDown(event):
4      lastAction[0] = 'left'
5      lastAction[1] = [event.Position[0],event.Position[1]]
6
7  hm = pyHook.HookManager()
8  hm.MouseLeftDown = OnMouseLeftDown
9  hm.HookMouse()
10 pythoncom.PumpMessages()
```

Figure 4.3: Using the pyHook module to keep track of left mouse button clicks.

4.5 Window Watcher

The goals for the visualisation tool *Window Watcher* are to visualise (1) spatiotemporal, (2) spatial and (3) temporal data aspects, as well as (4) user actions. These four goals are specific for the type of data *Window Watcher* is designed to analyse, i.e., window use data.

Spatial data is of interest, because screen real estate is precious and limited. Even with large multi-monitor environments, users can need more display space than there is available. To learn more about how users distribute windows across this space *Window Watcher* must, therefore, support spatial representations. Space in this context is a three dimensional concept. Windows are explicitly positioned in 2D space by the user by manipulating their position and size. Windows also move in depth, with most of this occurring implicitly as a side effect of bringing a window into focus: for each window brought to the foreground, several windows may be implicitly moved deeper on the z-axis.

Temporal representation of data is relevant as previous work suggests that 'window thrashing' occurs (Mackinlay and Royer, 2004), with users executing frenetic short term bursts of window management. This suggests that there is a strong temporal aspect to window use.

The following sections describe how *Window Watcher* visualises spatiotemporal, spatial and temporal window use data, as well as user actions. The examples are all actual data samples, gathered with *PyLogger* (note that colour is extensively used in *Window Watcher*, so many of the figures are unavoidably poor when viewed in grey-scale).

4.5.1 Spatiotemporal data

Window Watcher can replay logged events. This approach is similar to the study by Uehling and Wolf (1995) (who studied interaction with a particular application rather than with the screen 'as a whole', as is the case in the current study) . The 'playback' window provides a view of the full extent of the user's display space. During playback, the content of the logged user's display is continuously updated to reflect changes in their windowing state.

The size and shape of the playback window reflects the total amount of screen estate the user has available, i.e., a setup with two monitors with 1680×1050

pixels resolution placed side by side is shown as one 'unified' 3360×1050 pixels area in the playback window, albeit scaled down. Figure 4.4 shows such a user who has two screens, both with a resolution of 1680×1050 pixels. The logged user's main screen is on the left side, as indicated by the solid red bar along the bottom which represents the Windows Taskbar. All windows (and the Windows Taskbar) are shown in their actual locations.

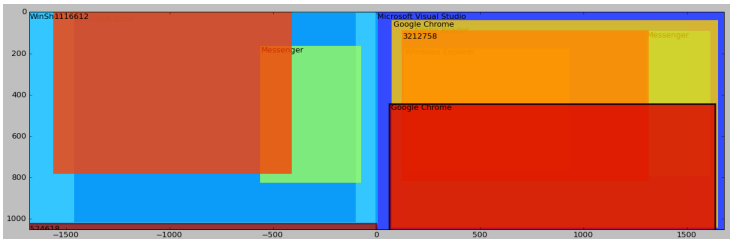


Figure 4.4: An example of the playback window.

Window z-order is essentially the 'depth' of windows on the screen. Z-order is a critical notion for some window switching tools: for instance, successive Alt+Tab presses traverse windows in their z-order. *Window Watcher* uses colour to display the window z-order. The top window is dark red (hot), and the window with the lowest z-order is dark blue (cold). The desktop background is displayed in white. The window that currently has focus is indicated by a black border.

The playback window also shows application names for most common applications (e.g., email clients, Microsoft Office tools, web browsers, etc.). For windows where *Window Watcher* is unable to identify the application the unique identification code for the window (the window handle) is shown.

Window use data can be played back in real-time, but for data files that span several weeks this is not a realistic option. Linear speed up is not feasible because, although long periods of inactivity can be compressed to a reasonable rate, periods of high activity are played back too rapidly to be of use. Therefore, with linear speed up of playback detailed information about the nature of window interaction can be lost, such as the burst-like characteristic of window use activities.

A time conversion mechanism is applied as follows: If the time between two events is small, the playback is slowed down, and if the time between two events is large the playback is sped up by an algorithmically determined factor. The algorithm converts the time between two events as follows: if the time t between

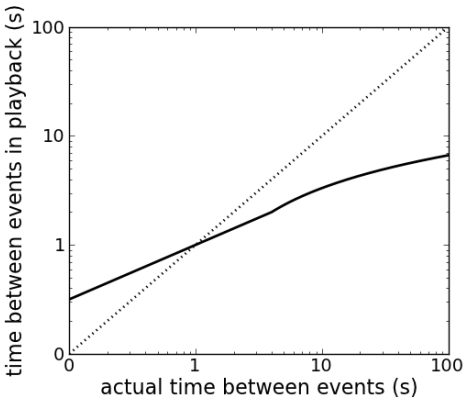
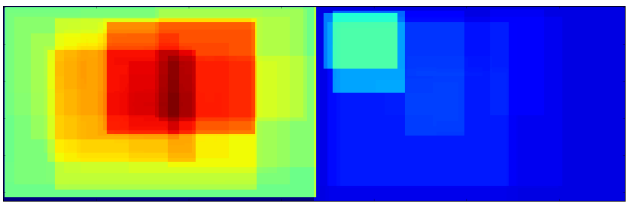
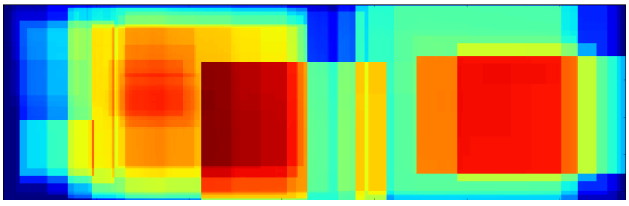


Figure 4.5: Actual time between events and the playback time in seconds using the conversion described in this chapter (dotted line is real time playback). Note both x and y scales are logarithmic.

two events is smaller than a lower bound (4 seconds) the playback time is the square root of t, if the time t between two events is larger than the lower bound the playback time is $log_2 t$. This conversion is shown in Figure 4.5. The conversion leads to the desired effect: slowing down small values and speeding up (very) large values. Optionally, this time transformation can be scaled by introducing a multiplication factor to speed up or slow down the replay.



(a)



(b)

Figure 4.6: Heatmaps for two different users.

4.5.2 *Spatial data*

Window Watcher keeps track of which parts of the screen are covered by a window and to what depth. This information is shown in a heatmap. The heatmap conveys the popularity of certain parts of the screen. Also, it can help to identify certain use patterns. For example, Figure 4.6 shows heatmaps of two days of data for two different users, both of whom use dual 1680×1050 monitors. The user shown in Figure 4.6a has a clear preference for the left screen, while the user in Figure 4.6b uses both more or less equally. Also, the user in Figure 4.6a often has a window maximised in the left screen, while the user in Figure 4.6b does not maximise windows often.

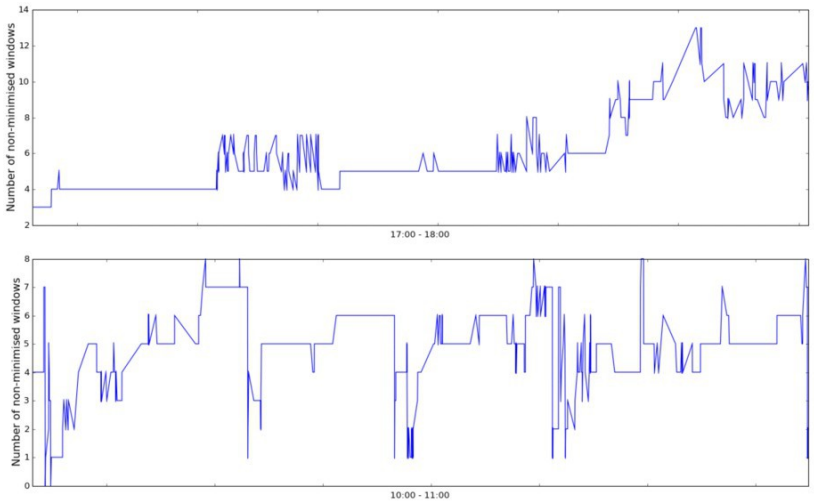


Figure 4.7: Two plots of the number of non-minimised windows over one hour.

4.5.3 *Temporal data*

An example of a temporal summarisation is shown in Figure 4.7. Here, the number of non-minimised windows over one hour is plotted for two different participants. The top plot shows gradual increases and decreases in the number of non-minimised windows, while the bottom plot shows sudden increases and decreases in the number of non-minimised windows. This temporal behavioural pattern is immediately visible when plotting the data over time, but might have been missed when, for example, merely looking at the average number of non-

minimised windows for each hour. In particular, these temporal plots visualise and help to identify episodes of window thrashing, described by Mackinlay and Royer (2004) as short periods of rapid window manipulation.

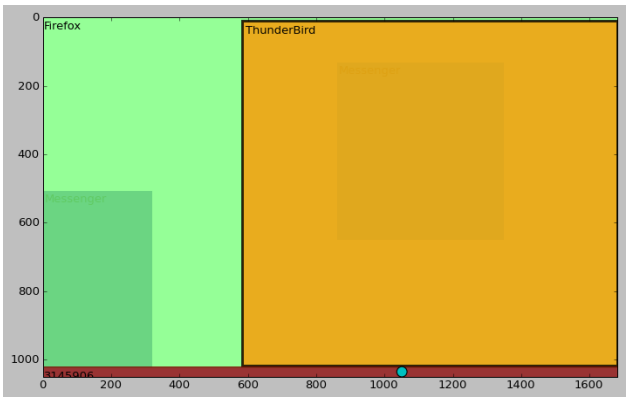


Figure 4.8: The location of mouse clicks is portrayed with a cyan circle, suggesting the method used to switch windows (in this case, the Windows Taskbar).

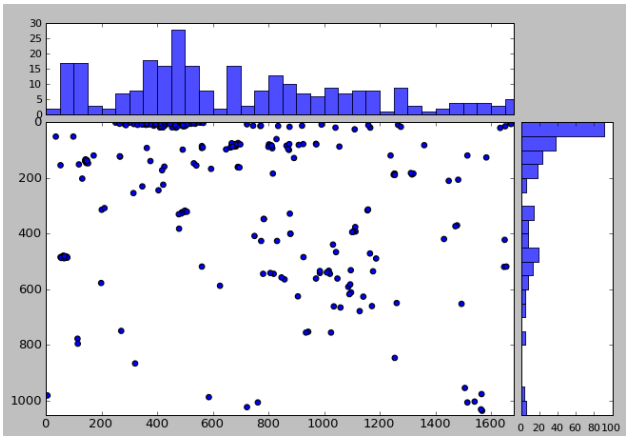


Figure 4.9: A scatter plot and two histograms visualising the locations of mouse clicks (Windows Taskbar clicks omitted).

4.5.4 User actions

Window Watcher portrays the method used to switch windows by using a cyan dot to display the location of mouse clicks during window switching. Figure 4.8, for

example, shows a snapshot of the playback window when the user switches to the Thunderbird window using the Windows Taskbar.

Figure 4.9 shows a summary statistics plot, revealing the location of a range of mouse clicks made by one user. The histograms show how many mouse clicks have occurred in the adjoining 'band' of pixels (vertical or horizontal). Clicks on the Taskbar have been removed (as this would lead to a strongly skewed histogram along the bottom x-axis). This visualisation, combined with the playback window, reveals an interesting behavioural pattern. Even when the user has a large portion of the Firefox window visible (see Figure 4.8), he/she often clicks on the title bar of the window to switch to this window (see Figure 4.8 and 4.9), even though a click anywhere in the window would bring the window to the focus.

4.6 Conclusion

For the design of novel interaction techniques information about current user behaviour is invaluable information. The description of *PyLogger* presented in this chapter should help and stimulate other researchers to develop similar software, which in turn can be used to gather additional information about the way users interact with their computer.

Statistical summaries, which can be easily extracted form the data generated by *PyLogger*, are useful, but it is the combination of quantitative statistics with visual replays that has proven most useful in the analysis of the data gathered in the *PyLogger* study (see Chapter 5).

Window Watcher is designed for, and only usable with, data generated by *PyLogger*. Plaisant (2004) observes that this phenomenon of visualisations being highly domain-specific is common. Nevertheless, some of *Window Watcher*'s design features, such as the time manipulation algorithm and heatmaps, will be generalisable to other domains as well, for example, scrolling (time manipulation) and menu use (heatmaps).

In general, *Window Watcher* has proven useful even when it was still in the very early (and not very sophisticated) phases of development. Even a very crude and 'sketchy' version of the 'playback' window in particular revealed many interesting avenues to explore, which had not made themselves apparent from the log data (a clear example of the "*answering questions you didn't know you had*"

noted by Plaisant, 2004, p. 111). Therefore, I feel it is advisable for any researcher working with similar user data to (at the very least) produce such a visualisation, which should not take much time or effort, in particular when care is taken that the data generated by the logging tool is of a usable format to be 'ported' to a visualisation.

Chapter V

An Empirical Characterisation of Window Use

To "know thy user" is a classic admonition in user interface design (Hansen, 1971; Preece et al., 2007). Having a good understanding of how users interact with windows is important for informing the design of new and improved window management tools. However, there have been relatively few empirical studies of window manipulation on common operating systems. Hutchings et al. (2004) and Hutchings and Stasko (2004a) studied window management and display organisation, respectively, but as these studies are several years old the results may no longer reflect current use. Three reasons for suspecting a change in behaviour from previous studies are the increased size and resolution of computer displays, the widespread use of multi-monitor environments, and the increased popularity of tabbed applications.

Chapter 4 introduced and described *PyLogger* and *Window Watcher*. A group of volunteers was asked to install *PyLogger* for three weeks. This chapter presents the findings of this study, categorised by six aspects of window use: (1) the number of open windows, including how many windows are non-minimised and visible, as well as how often applications have multiple windows associated with it, (2) window switching, including frequency of switching and the tools people use for window switching, (3) the efficiency and use of the three most common tools for switching between windows, (4) display space management, (5) revisitation to applications and windows and (6) window geometry management actions, such as moving and resizing.

5.1 Contributions and findings

- an empirical characterisation of interaction with windows;
- differences between single and dual monitor users in terms of the number of open, non-minimised and visible windows, and the discovery that participants had fewer windows open and visible than in previous studies;
- window switching is a very frequent activity;
- acquiring a particular window by navigating through application-grouped items on the Taskbar is slow, and Alt+Tab is seldom used for retrieving anything other than the most recently used window;
- an updated classification of stereotypical window management styles (pilers, maximisers, near maximisers and splatterers);
- strong application and window revisitation patterns;
- implications for the design of window switching tools.

5.2 Definitions

The term *display space management* captures the user's manipulation of the display space, including how many windows there are, what the window states are, and the location of windows on the screen (including position in the z-ordering). Display space management is usually a manual process, but automatically generated dialog boxes and pop-up windows are an exception. There are four window *states*: *minimised*, *fully visible*, *partially visible* and *hidden* (see Figure 5.1). All windows are either minimised or non-minimised (minimising a window is sometimes also referred to as *iconifying*). Non-minimised windows are either visible or hidden. Window (in)visibility has consequences for how the window can be reached. To make a hidden window the focal window a window switching interface, such as the Windows Taskbar or Alt+Tab *must* be used, while (partially) visible windows can be acquired by a mouse click in the window. It is not possible for the user to visually identify whether a window is minimised or hidden; in both

cases the window is not visible (*PyLogger* is able to differentiate between these states). However, it does have consequences for how the window can be reached: windows that are occluded by other windows can be reached by minimising or closing occluding windows, but minimised windows can only be reached by using a window switching interface.

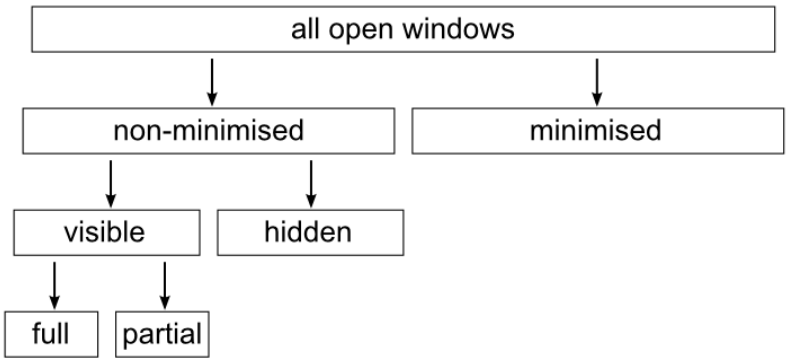


Figure 5.1: A classification of different window *states*.

5.3 Participants and procedure

Twenty-five people participated, all university students or employees. Age ranged from 21 to 61 years old, with a mean of 31 years old. Reported computer use ranged from 25 to 90 hours per week, with a mean of 49 hours. Two participants installed the software on two computers they regularly used; for the analysis the largest of the two data sets was used, i.e., the computer they used most. Seven participants used Windows Vista, the others used Windows XP. Nine users used a single monitor setup, nine users had a dual-monitor setup and seven users had a mix of both (e.g., a laptop that is sometimes extended with an extra monitor). However, of these users using a mix of monitor setups, five used a particular setup more than 80% of the time, and these users were classified accordingly.

Participants were able to stop the logging at any time and received instructions on how to do this. Also, participants were shown where log files were stored on their computer, and were told that they could removes any files at any time. Participants had the right to withdraw from the study at any time, including withdrawal

of any information provided (none chose to do so). After the logging period *PyLogger* was removed from the participants systems and participants completed a questionnaire about *PyLogger* and window use.

5.4 Number of windows

The more windows are open, the more problems can be expected with finding and switching to the correct window. Therefore, the first step of the analysis is to study the number of windows and their respective states. For example, a high number of open windows, but a low number of non-minimised windows might declutter the screen, but does not alleviate the problem of finding the correct window. Similarly, information about the number of visible windows is relevant, as it tells something about how 'crowded' the screen is. Lastly, as some window switching tools group windows by application, statistics about how many windows are open per application are informative.

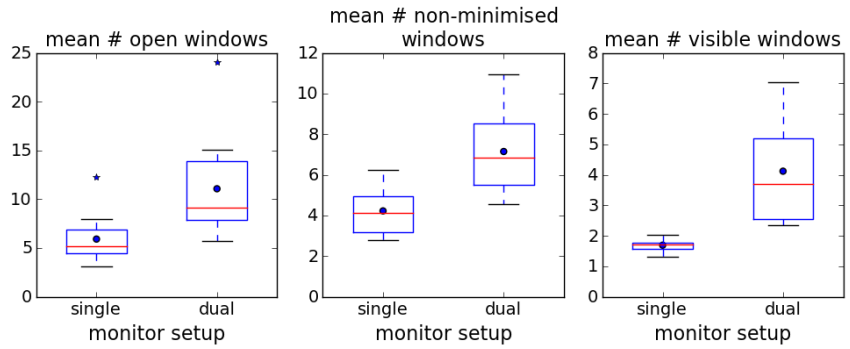


Figure 5.2: Boxplots of the number of open windows, the number of non-minimised windows and the number of visible windows, split by monitor setup.

5.4.1 Open windows

On average, users had 8.5 windows open (*SD*=4.6). However, such a 'compound' statistic is dependent on the ratio of single to dual monitors users in the study, as evident from the results found by Smith et al. (2003) and Hutchings et al. (2004).

Therefore, it is vital to analyse these two groups of users separately, as shown in Figure 5.2. Two things are apparent from Figure 5.2a: (1) dual monitor users have more windows open than single monitor users, (2) there are two 'extreme' users (one single monitor user and one dual monitor user) with an exceptionally high number of open windows. These two findings are examined in more detail in the following sections.

Single and dual monitor users

On average, single monitor users have 5.9 windows open (*SD*=2.4), while dual monitors users have 11.1 windows open (*SD*=4.9). This difference is significant ($p<.001$, Mann-Whitney U test). However, it is not clear whether the amount of screen estate also influences the number of windows open, or whether the key determinant of the number of open windows is the number of screens (regardless of the size of these screens). Figure 5.3 shows the relation between screen estate (expressed as the numbers of pixels) and the total number of open windows. Correlation analysis fails to find significant correlations between screen estate and the numbers of open windows for both single monitor users ($p=.3$) and dual monitor users ($p=.1$). It seems that while the number of screens does influence the number of windows a user has open, the size of these screens does not matter much. However, the absence of a significant correlation could be due to restriction of range; all users in the current study used a regular laptop computer or desktop setup, meaning that no users with very small or large screens were included.

Extreme values

Regarding the extreme values (data points larger than 3 *SD*'s above the mean) shown in Figure 5.2a, it is important to note that a high number of open windows does not necessarily mean the screen is cluttered. Figure 5.4 shows a representative *Window Watcher* (see Chapter 4) snapshot for the dual monitor user with a very high mean number of open windows (24). This user maintains a 'neat' (but deep) pile of windows, normally with the top one maximised, and uses one of the two monitors almost to the exclusion of the other. Even though this user has 29 windows open (and 11 non-minimised) at this particular moment, the screen does not look cluttered.

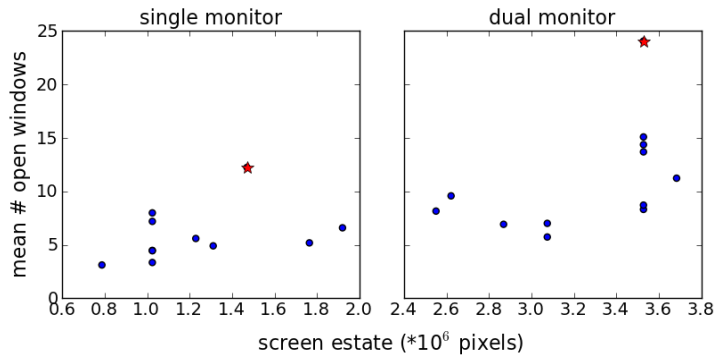


Figure 5.3: Scatterplots showing the total number of windows by screen estate, split by single and dual monitor use. Stars indicate the extreme values also shown in Figure 5.2a.

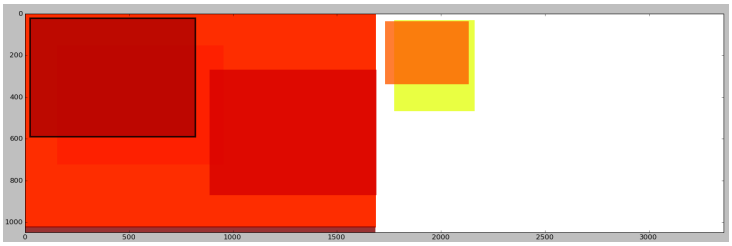


Figure 5.4: Playback window for a user with a very high number of open windows, but a maximised window obscuring them.

5.4.2 Non-minimised windows

Figure 5.2b show the number of non-minimised windows, split by monitor setup. Dual monitor users have more non-minimised windows than single monitor users; single monitor users have 4.2 non-minimised windows on average (SD=1.2), while dual monitors users have 7.2 non-minimised windows (SD=2.1). This difference is significant ($p<.001$, Mann-Whitney U test). These results are *partially* similar to the statistics regarding the total number of open windows, presented in the previous section: the results once again reveal a difference between single and dual monitor users, but the two 'extreme' users (in terms of the number of open windows) are no longer identifiable in the statistics regarding the number of non-minimised windows. The latter issue is explored further in the following section.

Extreme values

The two users with a very high number of windows open (see Figure 5.2a) do not have a very high number of non-minimised windows. This suggests that a 'coping strategy' when the number of open windows is high is to minimise many of these windows. Figure 5.5 shows this effect: the more windows a user has open, the smaller the proportion of windows that is non-minimised (r=-.6, p<.001).

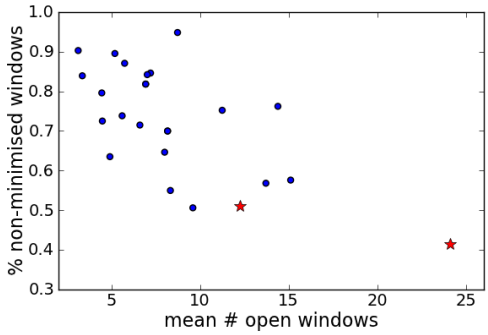


Figure 5.5: Scatterplot showing the percentage of non-minimised windows by total number of windows. Stars indicate the extreme values also shown in Figure 5.2a.

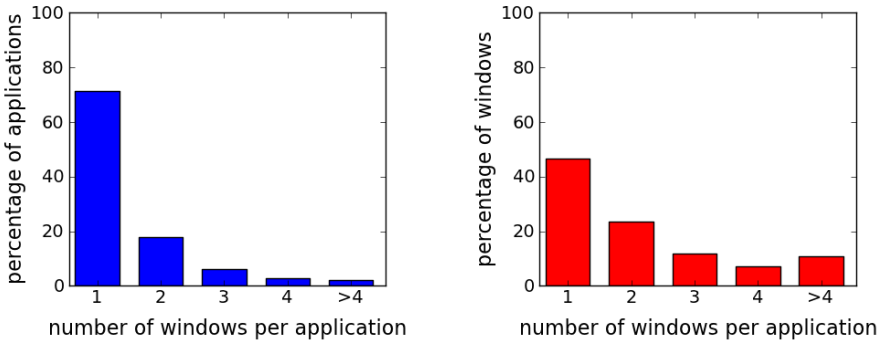
5.4.3 Visible windows

Figure 5.2c show the number of visible windows, split by monitor setup. Dual monitor users have more visible windows than single monitor users; single monitor users have 1.7 visible windows on average (*SD*=0.2), while dual monitors users have 4.1 visible windows (*SD*=1.5). This difference is significant (p<.001, Mann-Whitney U test).

5.4.4 Applications and windows

Several common window switching interfaces, including the Microsoft Windows Taskbar, group windows by application. This grouping has potential efficiency benefits when there are multiple windows associated with an application, because the user can focus his/her search for a specific window on the relevant application group. An analysis of how often there is more than one window associated with

an application reveals that 71.1% of open applications have only one window associated with it, or, vice versa, 28.9% of open applications have more than one window associated with it (also see Figure 5.6a). Put differently, 53.3% of windows belong to an application that has more than one window associated with it (also see Figure 5.6b).



(a) (b)

Figure 5.6: (a) Percentage of applications with one, two, three, four or more than four windows open and (b) percentage of windows that belong to an application with one, two, three, four or more than four windows open.

These results show that it can be beneficial to group window by application: more than half of the windows belong to an application that has more than one window associated with it. By grouping these windows together according to application they become easier to find as users can limit the search for the window to the region of the screen (or window switcher) where this particular application is located.

5.5 Window switching

There are three ways a new window can become the focal window, i.e., a window switch: a switch *between* (already open) windows, opening a new window and closing a window (causing the window below that window in the z-ordering to become the focal window). This section examines the frequency of window switching, as well as which methods are used for these three types of window switching. This information is important for the development of new window switching and

management tools. For example, is there a difference between how often dual and single monitor users switch between windows? Does the method people use to switch between windows relate to the number of windows they have open? Is application launching common, or are most new windows that are opened additional windows for an application that is already open?

5.5.1 Frequency

Across all users, a window switch takes place 515 times per day, on average. However, these window switches are far from uniformly distributed across the day. Figure 5.7 shows the cumulative frequencies for window activation times ranging from 0 to 60 seconds. All activation times smaller than 200ms were removed, as these are most likely artifacts of the logging procedure, or, if anything, too short to grab the attention of the user. The analysis reveals a median window activation time of 4.3 seconds, as indicated in Figure 5.7. This is similar to previous work which found that the average time any window is active is a 20.9 seconds and that the median activation time is 3.77 seconds (Hutchings et al., 2004).

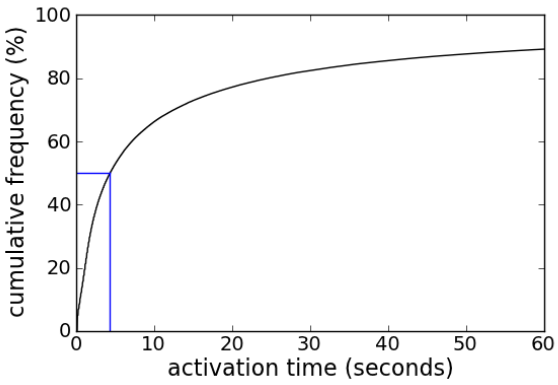


Figure 5.7: Cumulative frequencies of window activation times.

The 'compound' statistic of 515 switches per day is more informative when split by type of window switch; a switch *between* windows, opening a new window and closing a window. Switching between open windows is the most common type of window switch, accounting for 42% of window switches, and opening and closing window each account for 29% of window switches.

Single and dual monitor users

Results presented earlier in this chapter revealed large differences between single and dual monitor users regarding the number of open, non-minimised and visible windows. Therefore, this section analyses whether similar differences can be found in terms of the frequency of window switching. Figure 5.8 shows statistics regarding the number of switches between windows, the number of windows opened and the number of windows closed per day, split by monitor setup. Single monitor users switch between windows 150 times per day (*SD*=96), on average, dual monitor users 270 times per day (*SD*=114), which is a significant difference ($p<.01$, Mann-Whitney U test). The difference between single and dual monitor users in terms of opening and closing windows is less pronounced and not significant ($p=.1$), with single monitor user opening a window 134 times per day (*SD*=96), and dual monitor users 156 times per day (*SD*=47). Similarly, single monitor users close a window 136 times per day (*SD*=95), and dual monitor users 157 times per day (*SD*=53), the difference between which is not significant ($p=.2$).

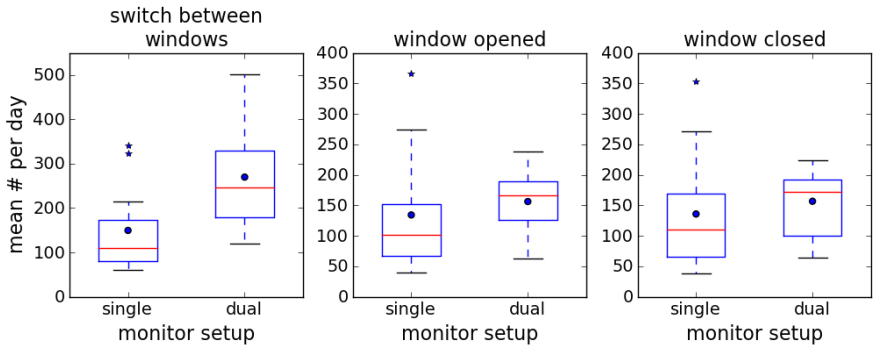


Figure 5.8: Boxplots of the number of switches between windows, the number of windows opened and the number of windows closed per day, split by monitor setup.

The results in the previous paragraph show that dual monitor users switch between windows more often than single monitor users. However, there could be a confounding factor as dual monitor users have more windows open, as shown in Figure 5.2a: possibly, the number of open windows, rather than the number of

monitors, influences how often users switch between windows Figure 5.9 shows the relationship between the mean total number of windows and the mean number of window switches per day, split by single and dual monitor use. Correlation analysis fails to find a significant correlation between the total number of windows and number of window switches for both single monitor users (p=.7) and dual monitor users (p=.7).

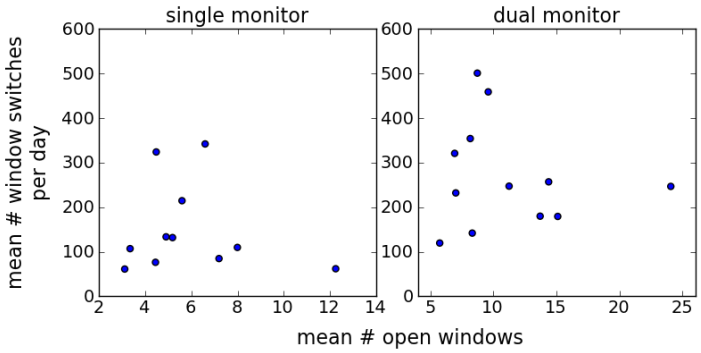


Figure 5.9: Scatterplots showing the mean number of visible of windows and the mean number of window switches per day, split by single and dual monitor use.

The results in section 5.4 showed that dual monitor users have more windows open, non-minimised and visible compared to single monitor users, but this does not explain why they also *switch* between these windows more often. Figure 5.9 suggests that the higher frequency of window switching for dual monitors users can not simply be attributed to these users having relatively more windows open. Rather, the main determinant seems to be the number of monitors. A possible explanation for the higher frequency of window switching found for dual monitor users is that they are more likely to multitask, as previous work has found that users of multi-monitor displays multitask more often than single display users (e.g., Truemper et al., 2008).

5.5.2 Methods used for switching between windows

Figure 5.10 shows which methods people use to switch between windows. In this section the use of the three most common methods are analysed in more detail.

Figure 5.10 suggests that the Windows Taskbar and a direct click are equally popular methods (38.9% and 36.8%, respectively), but some qualification is in

order. Similar to previous work (Hutchings et al., 2004), the results show that dual monitor users use a direct click more often than single monitor users, but use the Windows Taskbar less often (see Figure 5.11). The following paragraphs analyse the use of the three most used methods for switching between windows (a direct click, the Taskbar and Alt+Tab) in more detail.

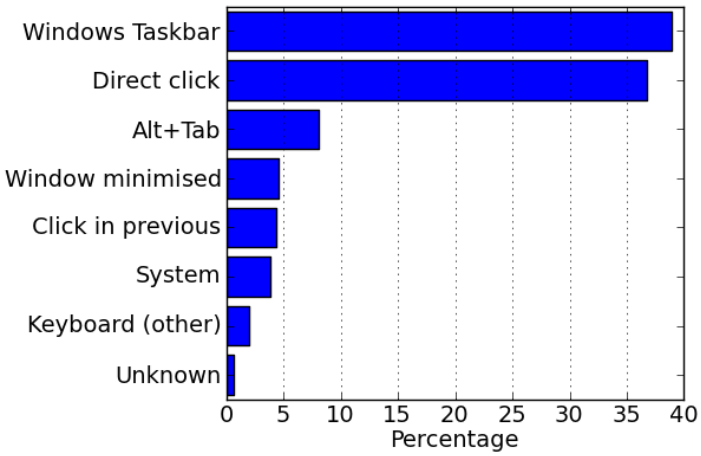


Figure 5.10: Methods participants used for switching between windows and their relative use. *Click in previous* implies a mouse click in the window that had focus before the window switch, e.g., clicking a hyperlink in an email message which causes a switch to the browser window.

Direct click

Similar to the previous section, which found that how often a user switches between windows is related to the number of monitors, but not to the number of windows, it is interesting to examine whether the discrepancy between single and dual monitor users in terms of their use of a direct click can *mainly* be attributed to the number of monitors, or whether the number of visible windows also plays a role. It is reasonable to suspect a relationship between the number of visible windows and the use of a direct click, as (partial) window visibility is required to be able to use a direct click. Figure 5.12 shows the relation between the number of visible windows and the use of a direct click. The results show that the more windows are visible, the more a direct click is used (r=.8, $p<.01$ for single monitor users and r=.6, $p<.05$ for dual monitor users).

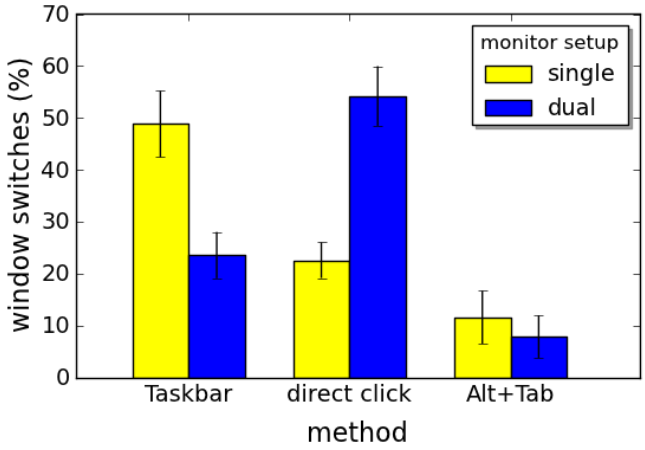


Figure 5.11: Relative use of a direct click, the Taskbar and Alt+Tab by single and dual monitor users. Error bars represent +/- 1 SE.

Taskbar

As a direct click is used more as there are more windows visible, it seems likely that the reverse is true for the use of the Taskbar. Figure 5.13 shows the relation between the number of visible windows and the use of the Taskbar. The results indeed show that the more windows are visible, the less the Taskbar is used (r=-.7, p<.05 for single monitor users and r=-.6, p<.05 for dual monitor users).

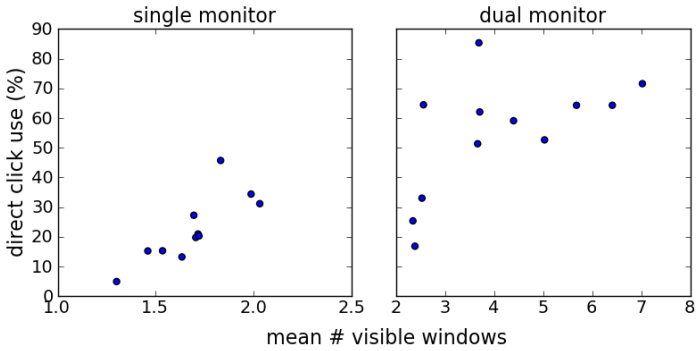


Figure 5.12: Scatterplots showing the number of visible of windows and the relative use of a direct click, split by single and dual monitor use.

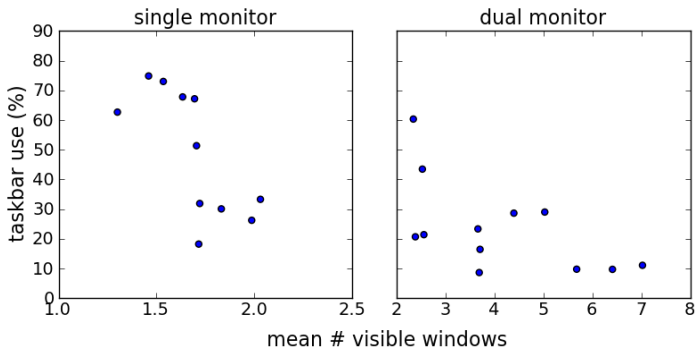


Figure 5.13: Scatterplots showing the number of visible of windows and the relative use of the Taskbar, split by single and dual monitor use.

Alt+Tab

The third most common method for switching between windows is Alt+Tab (8.1%). There is no difference between single and dual monitor users in terms of the use of Alt+Tab (see Figure 5.11). Whether someone is an Alt+Tab user or not (only participants who used Alt+Tab for more than 1% of their windows switches are considered to be Alt+Tab users) does not depend on the number of monitors he/she uses: 4 out of 11 single monitor users are Alt+Tab users, 6 out of 12 dual monitor users are Alt+Tab users. This contrasts the (informal) observation by Grudin (2001, p. 462) that dual monitor users "*[escape] from the need to Alt-Tab.*"

While previous studies have found a correlation between the number of hours per week someone uses a computer and the use of keyboard shortcuts (Peres et al., 2004, 2005), a similar analysis of the results from the current study does not find such a relationship for the use of Alt+Tab, with the mean number of hours per week someone uses a computer being 47 for Alt+Tab users, and 50 hours for users that do not use Alt+Tab.

Finally, it is possible that users with a high number of windows open are more likely to use Alt+Tab, as the two other main alternatives, a direct click and the Windows Taskbar, are less likely to be possible or practical when the number of windows is high (as it is less likely that the target window is visible, and the Taskbar can become overcrowded when the number of windows is high). However, Figure 5.14 shows that this is not the case; there is no correlation between the total number of window and the use of Alt+Tab (p=.9).

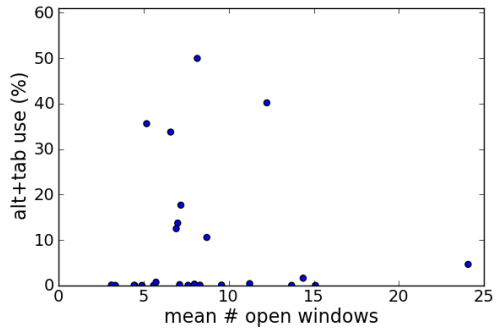


Figure 5.14: Scatterplot showing the total number of windows and the relative use of the Alt+Tab.

5.5.3 Methods used for opening windows

Figure 5.15 shows the relative use of various methods to open a new window. In this section, the three most common methods are analysed in more detail. Also, the results are analysed in terms of whether the new window was one for an already open application or whether a new application was launched.

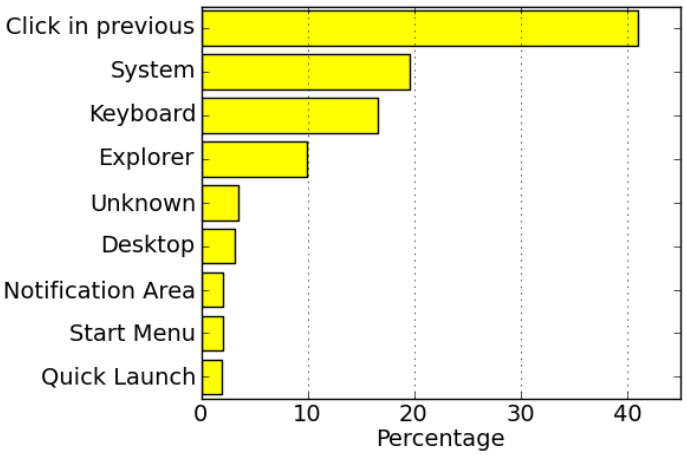


Figure 5.15: Methods participants used for opening windows and their relative use. *Click in previous* implies a mouse click in the window that had focus before the window switch, e.g., clicking a hyperlink in an email message resulting in a (new) browser window opening.

Click in previous window

The most common method for opening a new window (41.0%) is a click in the previous window (see Figure 5.15). Two examples of this are (1) clicking on a hyperlink or attachment in an e-mail message and (2) clicking on a '*mailto:*' hyperlink in a browser window.

System

The second most common cause for a new window opening is a system action (19.6%), i.e., not user-generated. This is a possible *interruption* for users, distracting them from their current task. Previous work has identified that programs such as email applications and instant messengers indeed commonly interrupt the user from his/her current task (Adamczyk and Bailey, 2004), and these interruptions are a widely studied research area (e.g., Adamczyk and Bailey, 2004; Bailey et al., 2001; Cutrell et al., 2000; Mark et al., 2005; Oulasvirta and Saariluoma, 2006). Iqbal and Horvitz (2007) provide a thorough field study of interruptions and resuming tasks, including the frequency of interruptions and how long it takes the user to return to the original task.

Keyboard

The third most common method for opening a new window is a keyboard action (16.6%). This may seem surprising as keyboard-based methods are relatively unpopular for switches between windows (see Figure 5.10). Analysis reveals that a prevalent case is opening a 'Find' dialog box using 'Ctrl' +'F'.

New window or new application?

In the majority of cases (81%) where the user opened a new window this was a new window for an application that was already active. In the remaining minority of cases a new application was launched. Therefore, the low percentages for the use of Start Menu and Quick Launch in Figure 5.15 are slightly misleading in the sense that they are an artifact of application launching being an infrequent task.

5.5.4 *Methods used for closing windows*

Figure 5.16 shows the relative use of various methods for closing windows. The most common method is a mouse click (59.7%) in the window, i.e., clicking on the 'x' button in the top right corner of the window. The second most common cause for a window closing is a system action (23.7%). Typical examples are 'copy' and 'sending message' (for e-mail applications) dialogs that automatically close when the action is completed.

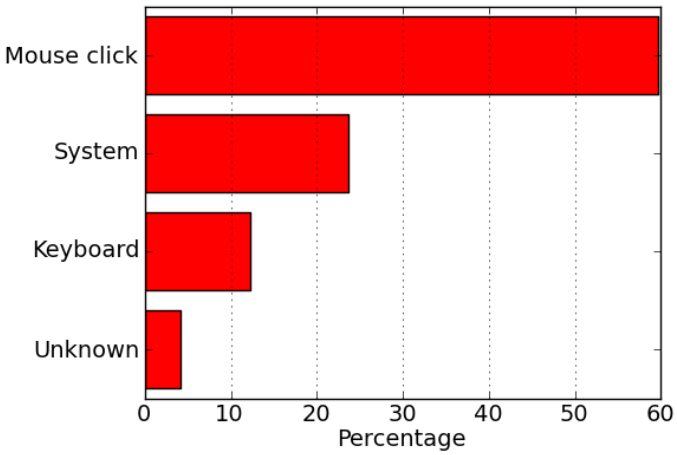


Figure 5.16: Methods participants used for closing windows and their relative use.

5.6 Tools for switching between windows

This section analyses some properties specific to the three most popular methods for switching between windows (the Taskbar, a direct click and Alt+Tab) in more detail. In particular, the efficiency of the various Taskbar buttons types (ungrouped, grouped and collapsed), window visibility when a direct click is used and the position of the target window in the Alt+Tab ordering are studied.

5.6.1 *Windows Taskbar*

The Windows Taskbar is prone to becoming overcrowded when the number of open windows is high. There is a mechanism in Microsoft Windows XP and Vista that automatically deals with this overcrowding; Taskbar button grouping.

Taskbar button grouping causes windows of the same application to always be next to each other on the Taskbar (see Figure 5.17b). This is in contrast to an *ungrouped* Taskbar, where Taskbar buttons are simply in the same order as the respective windows were opened (see Figure 5.17a). Furthermore, when Taskbar button grouping is enabled windows of the same application are *collapsed* under one application button when available space on the Taskbar becomes scarce (see Figure 5.17c). When windows are collapsed under an application button on the Taskbar two mouse clicks are required to switch to a window rather than one; a click on the application button on the Taskbar first, then selecting the window in a sub-menu.

(a) Ungrouped

(b) Grouped by application

(c) Collapsed by application

Figure 5.17: Three Windows XP Taskbar button states.

For each of the participants in the current study, it was noted whether Taskbar button grouping functionality was turned on or off. The majority of users had this option turned on (this is the default setting); only six participants had turned this option off. Some of the users who had turned off Taskbar button grouping expressed a very strong dislike of the grouping functionality of the Windows Taskbar in the post-study interview. These participants disliked the *collapsing* of Taskbar buttons in particular for two reasons: (1) it prohibits them seeing all available windows just by glancing at the Taskbar, and (2) two steps are required to switch to a window that is listed under a collapsed Taskbar button.

An analysis of the time a window had focus until a switch to another window has been completed using a Taskbar button reveals that the three different states of the Taskbar buttons (ungrouped, grouped or collapsed) affect window switching time. The average time a window has focus until a switch using an

ungrouped Taskbar button is made is 9.8 seconds (calculated using the geometric mean as described in Sauro and Lewis (2010) as the distribution of times is positively skewed). When a grouped, but non-collapsed button is used this time is 8.1 seconds. This suggests a beneficial effect of grouping Taskbar buttons of the same application; it is 1.7 seconds faster to switch to another window using a Taskbar button when it is grouped by application. However, this time advantage disappears when a collapsed Taskbar button is used; the average time a window has focus until a switch using a collapsed grouped Taskbar button is completed is 12.2 seconds. This is 2.4 seconds slower compared to the ungrouped Taskbar button, and 4.1 seconds slower compared to a grouped (but non-collapsed) Taskbar button. Statistical analysis reveals that all these differences are significant ($p<.001$, Kruskal-Wallis test with post-hoc pairwise comparisons). These results suggest that acquiring a particular window when it is listed under a collapsed grouped Taskbar button is slow (even slower than using an ungrouped Taskbar button), but that grouped non-collapsed Taskbar buttons do have an efficiency advantage.

5.6.2 Direct click

To be able to use a direct click to switch to a window the target window has to be at least partially visible. This section calculates how much of the target window was visible when a switch to it was made using a direct click on the window. This *visibility* is expressed as percentage; the part of the window that is visible divided by the total window size (see Figure 5.18). For the majority of switches (57.5%) the target window was not or hardly visible (0 to 10% visibility), meaning that a direct click is not possible or very improbable, as is indeed shown in Figure 5.19. However, even when the target window is (almost) completely visible (90 to 100% visibility) another method rather than a direct click is still used for 26.4% of window switches.

5.6.3 Windows Alt+Tab

Alt+Tab is a key combination for window switching. By pressing and holding down the 'Alt' key and (repeatedly) pressing the 'Tab' key the user can sequentially step through a list of available windows. Releasing the Alt+Tab switches to the window selected at that moment. The ordering of the list of windows (or

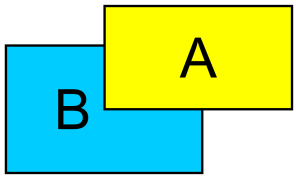


Figure 5.18: An example of window visibility: window A is 100% visible, window B is approximately 75% visible.

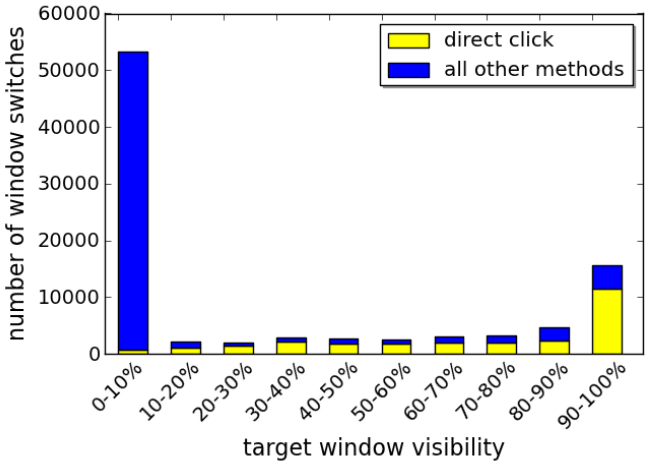


Figure 5.19: Target window visibility, direct click versus all other methods.

their representations) when Alt+Tab is used is based on the z-ordering of windows (for more details, see Section 2.1.1). This section analyses the use of Alt+Tab in terms of the position of the target window in the Alt+Tab ordering. Note that the position is always in the range [1, number of windows] (position 0 is the window that has focus).

For 63% of all window switches the target item is on position 1 in the z-ordered list of windows, and for 18% of switches it is on position 2. This means that in all, the target window is likely to be near the front of the Alt+Tab list; the mean position of the target item in the Alt+Tab list is 1.9. The mean position of the target window when Alt+Tab is used is 1.25, which shows that Alt+Tab is mainly used for switching to the most recently used window, not traversing much further along the list of windows (also see Figure 5.20). As across all users and all methods for switching between windows the average position of the target window in the z-

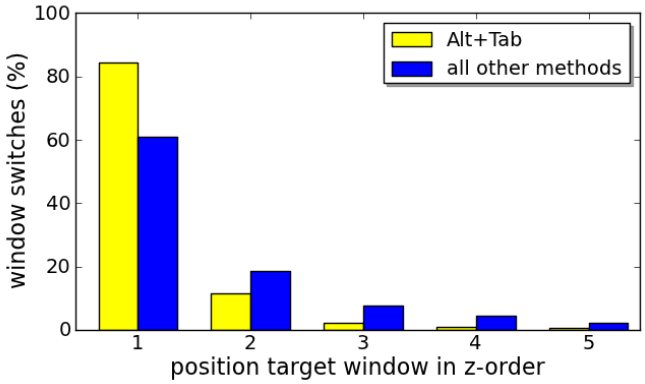


Figure 5.20: Position of the target window in the z-ordering of windows, Alt+Tab versus all other methods.

ordering is 1.9 this suggests that Alt+Tab could be an attractive option for almost *any* window switch. However, people mainly use it to just go back to the most recently used window, not traversing any further down the list of windows.

Possibly, people find it too difficult to select windows further down the Alt+Tab list as (1) a (long) sequence of key presses is needed, which users might find cumbersome, and (2) there is a risk of 'overshooting' the target window (which then needs to be corrected by stepping backwards through the list by using the key combination Shift+Alt+Tab). Window Vista addresses these problems by introducing the possibility to select a window in Alt+Tab by clicking on the icon/thumbnail using the mouse. Out of the seven Windows Vista users in the current study two were Alt+Tab users. These users predominantly used the keyboard to select the target window in Alt+Tab, not the mouse (they use the keyboard 99.8% and 95.7% of the time, respectively).

Another possible explanation for the under-utilisation of Alt+Tab for selecting a window at position 2 or further is that people might find it hard to anticipate or understand the order of items in the Alt+Tab window. While it is relatively easy to remember which window you used most recently, and will therefore be at position 1 in the Alt+Tab ordering (unless the window has been minimised), it is is increasingly harder to remember which window you have used second to last, third to last, fourth to last, and so forth.

5.7 Display space management

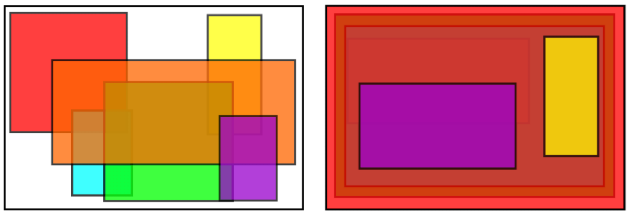
The statistics regarding the number of open, non-minimised and visible windows in section 5.4 do not tell the full story. Informal replay of the data using *Window Watcher* (see Chapter 4) revealed several distinct *window management styles*. To capture this difference in how users distribute windows across the screen, this section analyses the different window management styles of users by introducing a novel classification of window management styles, which is an extension of a previous classification by Hutchings and Stasko (2004a). Also, the amount of empty space, i.e., space that is not covered by any window, is examined in more detail.

5.7.1 Window management styles

As noted in the Section 3.1.2, the study by Hutchings and Stasko (2004a) identifies three different window management styles: *maximisers*, who maximise most of their windows, *near maximisers*, who manually resize window to almost full size, while leaving some space unoccupied for either the desktop or smaller windows such as instant messaging systems, and *careful coordinators*, who have many windows visible simultaneously and hardly ever maximise windows.

Such a taxonomy of window management styles can help to inform the future design of window management and switching tools by (1) guiding the development of specific tools to support these styles or (2) helping designers to ensure that novel tools are at the very least compatible with these styles.

However, Figure 5.21 illustrates some of the shortcomings of the classification by Hutchings and Stasko (2004a).



(a) Many windows visible (b) Many windows occluded

Figure 5.21: Two abstract examples of different window management styles.

Figure 5.21 shows two different management styles that, following the definitions in Hutchings and Stasko (2004a), would be problematic to classify, while analysis of the data from the current study (using *Window Watcher*) reveals that these two styles are common. The user in Figure 5.21a 'splatters' the windows all over the screen and many windows are (partially) visible, while the user in Figure 5.21b 'piles' windows on top of each other. The definitions by Hutchings and Stasko (2004a) are problematic for the classification of these two styles for two reasons. First, the style portrayed in Figure 5.21a would qualify as *careful coordinator* following the definitions by Hutchings and Stasko (2004a), but this 'splattering' style seems too 'haphazard' to really be appropriately described as *careful coordinating*, which seems to suggest some sense of careful organisation. Second, the style portrayed in Figure 5.21b would qualify as none of the styles as defined in Hutchings and Stasko (2004a); only one window is maximised (ruling out *maximiser*), the user leaves no space unoccupied for the desktop or smaller windows (ruling out *near maximiser*), but 50% of windows is not visible (ruling out *careful coordinator*).

The observations that (1) data about how windows are arranged on the screen provides information about users that window statistics alone can not convey and (2) the classification of window management styles by Hutchings and Stasko (2004a) is in need of some updating to fully capture the observed common window management styles in the current study inspired modifications of the styles identified by Hutchings and Stasko (2004a).

The goals of this updated classification are to provide mutually exclusive and exhaustive categories that are able to discriminate between the two common window management styles shown in Figure 5.21. The classification retains the *maximisers* and *near maximisers* categories from the study by Hutchings and Stasko (2004a), but replaces the problematic *careful coordinator* style with a *piler* and a *splatterer* style, as follows:

- **Piler**: Most (>50%) non-minimised windows are not visible (as these are completely occluded by other windows).

- **Splatterer**: Most (>50%) non-minimised windows are (partly) visible.

- **Maximiser**: A subset of pilers with the one window visible on the screen, while leaving hardly any or no space on the desktop visible (<10%).

- **Near maximiser**: A subset of pilers with the one window visible on the screen, while leaving some space on the desktop visible (>10%).

The following sections present the results of the analysis of window management styles for single and dual monitor users. All data is analysed on a per day basis: if the results show a user to use a certain window management style 75% of the time, it means that the user was recorded using this style 75% of the days that data was logged.

Single monitor users

Figure 5.22 shows that most single monitor user have a clear preference for one management style, and in most cases this is piling (maximising and near maximising are specific types of piling, as described above). This is not surprising, as single monitors do not lend themselves very well to splattering, as screen estate is limited. Only participants 3 and 5 do not have a clear preference for the piling style.

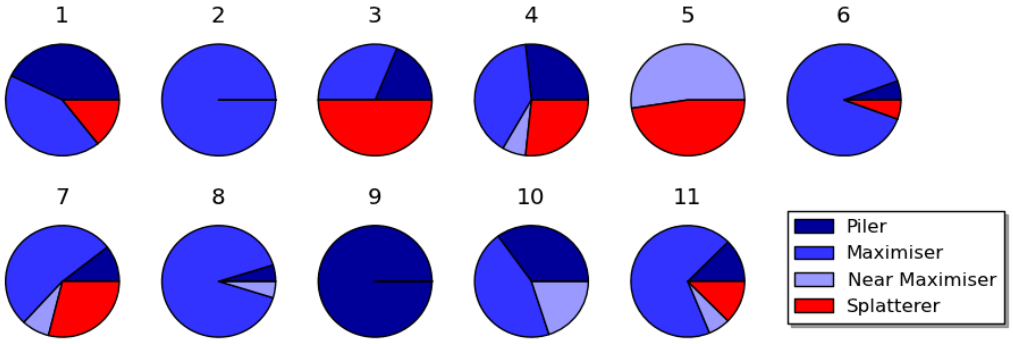


Figure 5.22: Observed window management styles for all single monitor users in the study, analysed per day.

Dual monitors users

As (1) Grudin (2001) observed that dual monitors users hardly ever straddle windows across monitors, and (2) it is possible that dual monitor users employ a different window management style on each monitor, the window management practices of dual monitor users were analysed on a 'per monitor' basis.

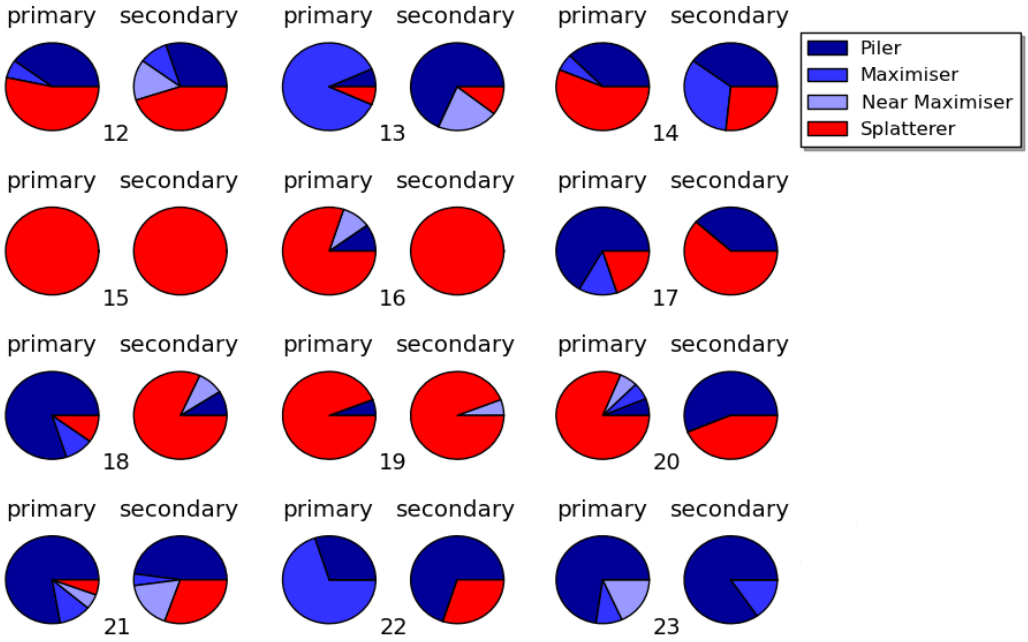


Figure 5.23: Observed window management styles for all dual monitor users in the study, analysed per day and split by primary and secondary monitor (the primary monitor is defined as the monitor that shows the Windows Taskbar).

Figure 5.23 shows the results of the analysis of window management style for dual monitor users. From Figure 5.23 several observations can be made. First, similar to single monitor users, dual monitor users often have a clear preference for one management style on each monitor, but this can be different between monitors. For example, participant 18 prefers piling windows on the primary screen, but splatters windows on the secondary screen (see Figure 5.24). This behaviour seems to reflect the informal remark made by a dual monitor user in the study by Bi and Balakrishnan (2009, p. 1010): "*I just throw all the non-primary applications to the secondary monitor, and do not care much about the layout.*" Second,

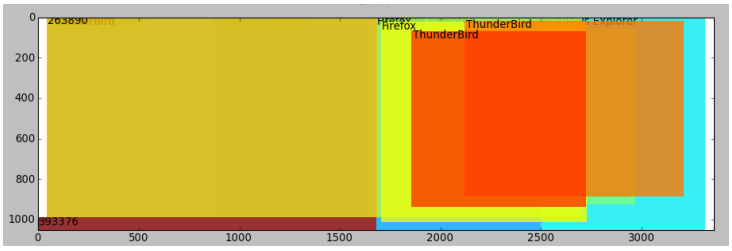


Figure 5.24: A representative window layout for user 18.

dual monitor users are rarely maximisers. Third, there are three dual monitor users (15, 16 and 19) who *splatter* their windows on both monitors. These three users all had two identical monitors, set to identical resolutions. Figure 5.25 shows a representative screen layout for user 15. Previous work (Grudin, 2001) suggests that a dual monitor setup is rarely or never treated as one large screen (i.e., the monitor serve clearly different purposes and windows are never straddled across screen), and it seems that these 'dual monitor splatterers' conform to this.

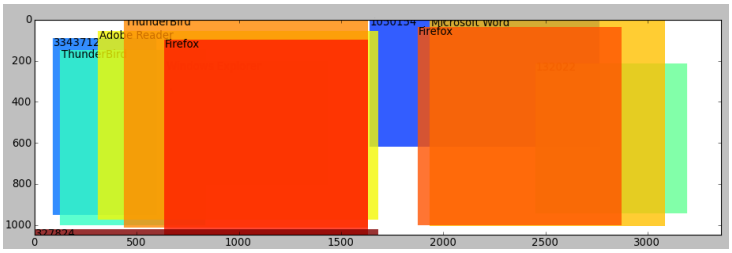


Figure 5.25: A representative window layout for user 15.

5.7.2 Empty space

Screen estate that is not covered by any window is *empty space*. Empty space reveals the *desktop*, which often contains (shortcuts to) applications and documents. By keeping a part of the desktop visible these items are easily accessible. Also, some new window management techniques rely on the availability of empty space (e.g., Hutchings and Stasko, 2002). Therefore, it is interesting to analyse the amount of empty space users keep.

Overall, both single and dual monitor users do not have a large amount of empty space (5.8% for single monitor users and 14.9% for dual monitor users),

but there are some differences. While the mean amount of empty space is not very different for single and dual monitor users, single monitor users have no empty space at all 88.4% of the time, but dual monitor users only 20.9% of the time. Additional analysis of the total time that less than 20% of total available space is empty (similar to the analysis in Hutchings et al. (2004)) reveals that dual monitor users do not leave a large portion of space uncovered, as 71.7% of the time less than 20% of available space is empty.

5.8 Revisitation patterns

This section presents data describing revisitation patterns, both to windows and to applications. Strong revisitation patterns have been identified in several areas of computer use (e.g., Alexander and Cockburn, 2008; Tauscher and Greenberg, 1997; Greenberg and Witten, 1993; Cockburn et al., 2007), and interfaces that explicitly support this revisitation have proven very succesful (e.g., Alexander et al., 2009). Therefore, if revisitation to applications and windows is common, this is an interesting avenue to explore for the development of window management tools.

5.8.1 Window revisitation

This section studies whether some windows are revisited more often than others, and how common revisiting recently used windows is.

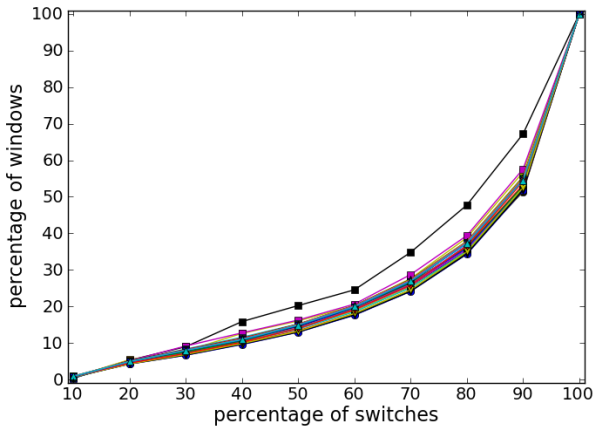


Figure 5.26: Percentage of window switches to percentage of windows for each participant, showing strong window revisitation.

Frequency

The average number of distinct windows per day ranges from 6 to 847 and has a cross-participant mean of 143 (*SD*=67). It is important to note however that many of these are dialog boxes and (temporary) pop-up windows; which are of a highly transient nature and, therefore, revisitation to these is therefore extremely rare. When dialog boxes and (temporary) pop-up windows are excluded from the analysis the average number of distinct windows per day ranges from 4 to 606 with a cross-participant mean of 59 (*SD*=32).

Figure 5.27: Visualisation of window revisitation; two windows are revisited very often, but many windows are hardly ever revisited.

An analysis of how often participants revisited windows per day, i.e., returning to a window that is already open, finds that 55% of windows (excluding dialog boxes and pop-up windows) are never revisited, i.e., they are opened, used, and closed again. A typical example of this type of window is an e-mail composition window. The analysis of window that are revisited at least once shows frequent revisitation; 80% of switches between windows is to 37% of windows (cross-participant mean). Figure 5.26 shows that this pattern of revisitation is very consistent across participants.

A visualisation of window switching makes these revisitation patterns clearly

visible. Figure 5.27 shows a visualisation[1] of a representative day of switches between windows of an average (in terms of switches per day and number of open windows) user in a graph structure. The vertices are windows labelled by their window handle, which is a unique identification number. The (font) size of the vertices represents how often the window was switched to/from. The edges represent switches between windows, the thicker the edge the more the user switched from window *A* to window *B* (note that the edges are *directional*). In Figure 5.27 it is immediately apparent that a few windows are revisited a lot, but many are rarely revisited (Figure 5.27 excludes windows that are *never* revisited).

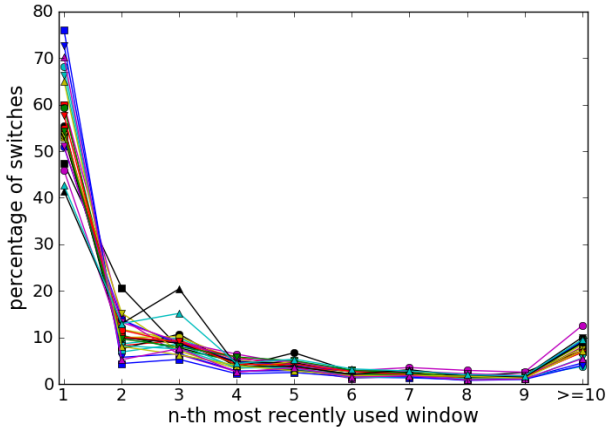


Figure 5.28: Percentage of switches sorted by recency of the target window, for each participant.

Recency

On average, 57% of switches (*SD*=9%) is to the most recently used window. Across participants this percentage ranges from 41% to 76% (also see Figure 5.28).

5.8.2 Application revisitation

This section describes long term revisitation patterns to applications. While many windows are transient, existing only for immediate work requirements (such as

[1] Visualisation generated using IBM Research's Many Eyes application: `http://www-958.ibm.com/software/data/cognos/manyeyes/`

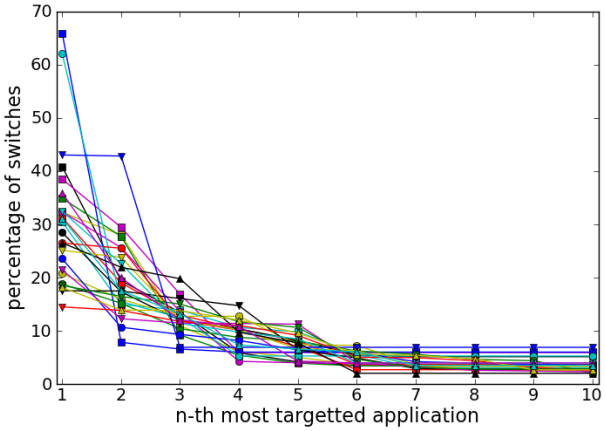


Figure 5.29: Percentage of switches to the ten most frequently switched to applications for each participant.

a window containing an email message during its composition) applications are relatively stable. For many tasks, such as 'check my email inbox', the user's target is likely to be the application (e.g., Outlook) rather than a specific window, so an application-based analysis is potentially informative.

Therefore, this section analyses revisitation to applications for the duration of the study. For each participant the total number of switches to the ten most switched to applications was calculated, ranked by frequency, and converted to a percentage of their total application switches. Figure 5.29 shows this data for each participant. It is clear that a few applications are switched to a lot, and a lot are used relatively little.

When this data is converted to Pareto principle data, similar to the analysis in the previous section, the results show that 80% of switches is to 2 to 11 applications, with a cross participant mean of 6 applications.

Analysis of the most revisited application per participant reveals that these are a browser (12 participants), e-mail client (4), IDE (3), file explorer (2), document editor (2), file sharing application (1), and a graphics editor (1). This is similar to previous work, which found that Internet applications grab 63% of the users' time (Beauvisage, 2009), and the majority of that time is spent using a web browser (Beauvisage, 2009; Bi and Balakrishnan, 2009).

5.9 Window geometry management

This section analyses the window geometry management actions by users. Window geometry management comprises of moving window and resizing windows. A previous study by Gaylin (1986) showed that window geometry management is far less frequent than window switching. As displays are getting larger and multi-monitor setups are becoming increasingly more popular, one could expect that window geometry management becomes more common, as people have more space available, and therefore might spend more time arranging their windows across this space.

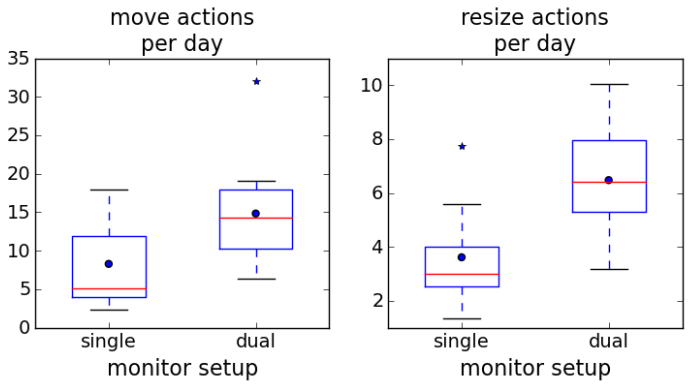


Figure 5.30: Boxplots showing the number of manual move and resize actions per day, split by monitor setup.

Figure 5.30 shows statistics regarding *manual* move and resize actions, not those automatically done by the system. Though dual monitor users move and resize windows more often than single monitor users, window geometry management is, in general, quite uncommon, similar to the results in Gaylin (1986). Single monitor users, on average, move a window 8.3 times per day (*SD*=5.4), and resize a window 3.6 times per day (*SD*=1.7). For dual monitor users these statistics are 14.8 (*SD*=6.7) and 6.5 (*SD*=2.1), respectively.

For an explanation of why users do not move and resize their windows very often, an analogy with the organisation of 'actual' desktops can be drawn. Malone (1983) studied the organisation of (office) desks, and were able to clearly identify 'neat' and 'messy' organisational styles. In the current study, there are no participants that seem to be particularly 'neat' and spend a lot of time managing

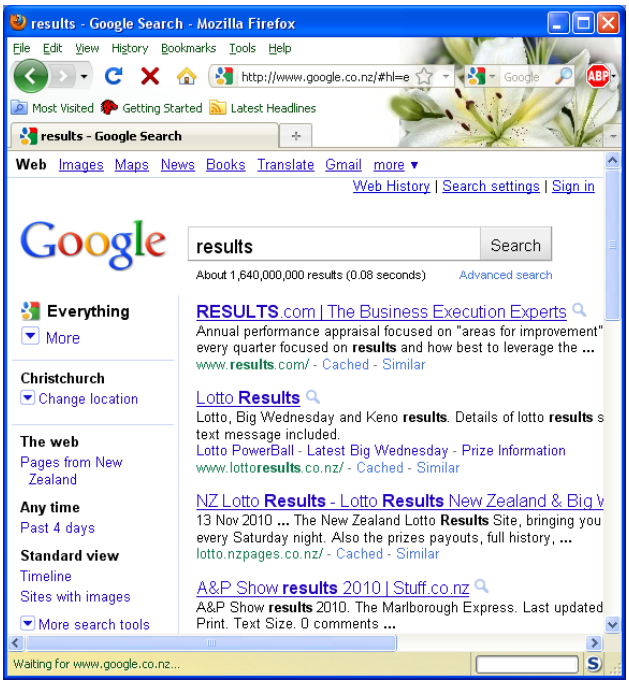


Figure 5.31: A browser window with poor usability due to the horizontal scrollbar at the bottom.

the place and location of their windows. There are several plausible explanations of why people do not engage in window geometry management often. First, there is not much additional benefit in window geometry management. Unlike the 'normal' desk in your office, where the organisation has long-term effects and benefits, window geometry parameters will be lost every time you restart your computer. Therefore, investing (a lot) of time in window management could be seen as a waste of time and effort. Second, there are only limited benefits to be gained from window geometry management, in particular resizing your windows. Unlike an office desk, where the amount of space to stack, pile and organise is abundant, the computer desktop has limited space available. Furthermore, many applications need a minimum amount space to be usable. For example, a browser window showing a Google results page will get a horizontal scrollbar if it is not at least 640 pixels wide, severely affecting readability of the results, and usability in general (see Figure 5.31). Therefore, although windows can theoretically be resized to *any* size, only a subset of those sizes are actually practical. Third, some researchers (e.g., Ahlström et al., 2009) have suggested that window geom-

etry management, and resizing windows in particular, is too error-prone, as the window border that has to be 'grabbed' for resizing actions is quite narrow. A few efforts have been made to improve window manipulation techniques. Ahlström et al. (2009) investigated several novel resizing techniques including a border-crossing technique where the user does not have the 'grab' the small border to select it, but moves the mouse across the border to automatically 'latch on' (also see Accot and Zhai, 2002) and a technique using a scaled down proxy window (with the same aspect ratio as the actual window). Ahlström et al. (2009) found that these new resizing techniques were all less error-prone than the traditional click-and-drag method.

5.10 Comparison to previous studies

It is interesting to compare the results from the current study to the results found in previous studies, because the most recent empirical studies of window use and management are several years old. The most comprehensive study of window management is by Hutchings et al. (2004). Hutchings et al. (2004) found that single monitor users on average have 3.5 windows visible on the screen, users of small multiple monitor setups (less than 3 million pixels of screen estate) 4.1 windows, and users of large multiple monitor setups (more than 3 million pixels of screen estate) 6.8 windows. Using the same definitions, the current study finds 1.7, 2.4 and 4.7 visible windows for these groups of users, respectively. Furthermore, Hutchings et al. (2004) find that 78.1% of the time users have eight or more windows open. Similar analysis of the results in current study reveals the participants have eight or more windows open only 47.1% of the time. In terms on the time users have no empty space visible most of the results in the current study are similar to those found in Hutchings et al. (2004), with exception of the amount of time single monitor users have no empty space. While Hutchings et al. find that single monitor users have no empty space 48.0% of the time, in the current study this is 88.4%.

Overall, there are several large differences between the results in the current study and those reported in Hutchings et al. (2004). Summarizing, the current study finds a lower number of open windows and a lower number of visible windows for all users, and significantly less time with no empty space for single

monitors users. The difference cannot be adequately explained by different participants' demographics; in the study by Hutchings et al. participants were all researchers in a sub-discipline of Computer Science, and in the current study almost all participants were either university students or employees in Computer Science as well.

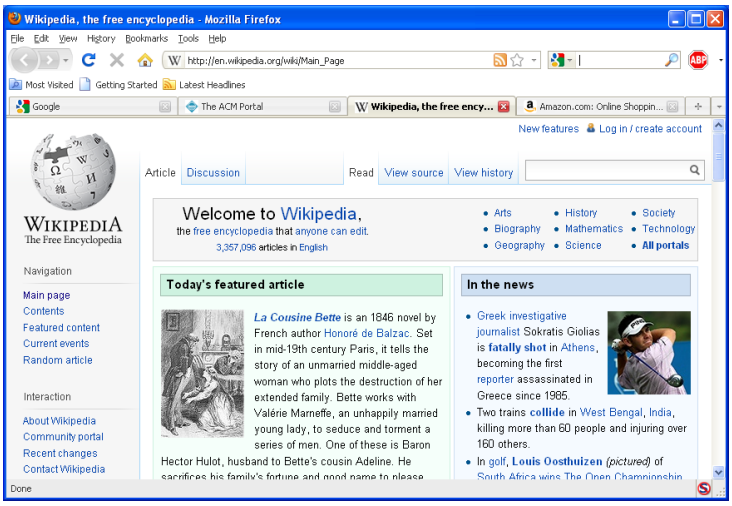


Figure 5.32: Mozilla's Firefox web browser with four tabs open.

One possible explanation for the discrepancy is the increased popularity of tabbed applications (see Figure 5.32). Tabbed interfaces eliminate the need to have multiple windows open to, for example, have two web pages open concurrently. A recent study (Dubroy and Balakrishnan, 2010) found a strong preference for the use of tabs over separate windows among Mozilla Firefox users. The Mozilla Firefox web browser made opening a link in a new tab rather than a new window the default behaviour with the introduction of Mozilla Firefox 2 in 2006, and Windows Internet Explorer introduced tabbed browsing in version 7, also released in 2006. Another explanation for the difference between the results found in the current study and Hutchings et al. (2004) is the increased multi-functionality of applications. For example, a web browser is no longer exclusively used for surfing the web, but also for instant messaging (e.g., through facebook.com or gmail.com) and even document creation and editing (e.g., through Google Docs).

5.11 Implications for the design of window switching tools

5.11.1 Number of windows

People have fewer windows open nowadays compared to a few years ago, which can most probably be attributed to the increased popularity of tabbed applications. However, this does not mean support for window switching is trivial or not important. Computer users have a large number of windows open, ranging up to a mean number of 24 open windows for one participant in the current study. The number of window switches exceeds 500 per day for several participants in the current study. Also, the results show that dual monitor users have more windows open and visible, and switch between windows more often than single monitor users. With the increased popularity of multi-monitor setups this means that good support for window switching becomes increasingly more important.

5.11.2 Window management styles

Piling and splattering are two very different window management styles. Potentially, different window management tools and window switching techniques might be more suited for one style or the other. For example, splatterers might benefit from more support for tiling windows. Splatterers have many windows visible simultaneously, but not all windows necessarily have a large portion visible. For splatterers, support for tiling might help to de-clutter their workspace. Pilers rely more on window switching tools like the Windows Taskbar and Alt+Tab, and might benefit from having more information about the position of (obscured) windows in the *z*-ordering.

5.11.3 Grouping by application

The results in the current study suggest that it is beneficial to group windows of the same application in a window switching interface. However, when this grouping implies that window switching now involves an extra step (in the case of the Windows Taskbar, first selecting the application button) performance benefits of grouping windows by application disappear completely and window switching becomes even slower compared to the ungrouped counterpart.

With the 'heaviest' collapsed Taskbar button user in the current study using a collapsed Taskbar button 40 times a day the inefficiency stemming from the collapsing technique might seem like a trivial problem. However, with the introduction of Windows 7 *all* Taskbar buttons are always collapsed by application by default. With the number of window switches using the Taskbar running into several hundreds per day this could mean a large productivity loss and potential source of frustration.

5.11.4 Z-ordering

The results regarding the low position of the target window in the z-ordering suggest that Alt+Tab could be an attractive method for nearly every window switch. Nevertheless, users only seem to use it for switching back to the previous window, not traversing any further down the list of windows. Possibly, people do not understand the ordering of windows in the Alt+Tab window, or, if they do understand, making rapid predictions about the position of the window takes too much cognitive effort. Potential solutions are visual aids to enhance knowledge of windows' positions in the z-ordering or a different ordering of windows in the Alt+Tab window.

5.11.5 Support for revisitation

The revisitation patterns observed in the current study suggest that supporting revisitation should be a design consideration for windows switching tools. More specifically, this support should allow users to quickly identify previously and commonly revisited windows in a switcher's display. Unfortunately, none of the current standard window switching tools explicitly support revisitation.

5.12 Conclusion

The study presented in this chapter gained valuable information about window use, including how people organise windows on the screen and when and how various window switching tools are used. These findings have important implications for the design of window switching interfaces.

Chapter VI

Supporting Window Switching with SCOTZ: Design and Implementation

The results presented in Chapter 5 showed that users often have many windows open and that switching between these windows is a very frequent task. It also identified several problems with current window switching interfaces, such as the finding that acquiring a particular window by navigating through application-grouped items in the Taskbar is slow. In general, current interface methods for window switching have changed relatively little since early graphical user interfaces: clicking on a window brings it into focus, as does selecting the window from a spatial iconic representation (e.g., the Windows Taskbar) or from a recency list (e.g., the Windows Alt+Tab display).

This chapter describes the design of a new window switcher called SCOTZ (for Spatially Consistent Thumbnail Zones). The first key design objective of SCOTZ is to provide a spatially stable layout of applications and windows. By providing a spatially stable layout users can make rapid decisions about the location of windows in the switching interface, reducing or even eliminating the need for visual search. Stability of window placement is achieved by grouping windows by application in spatially stable application zones (i.e., the application zones remain in stable positions, even when the computer has been restarted).

The second key design objective of SCOTZ is to support revisitation to frequently and recently accessed windows. The log analysis of real window and application switching presented in Chapter 5 showed that revisitation (returning to previously and/or often used windows and applications) is a frequent activity in computer use. Therefore, SCOTZ aims to improve the efficiency of switching to frequently and recently accessed windows. Switching to frequently accessed windows is supported by assigning more space to the most frequently switched to applications, thereby making them easier to find and quicker to select, and recently used windows are quickly accessible by retaining the functionality of Alt+Tab.

Additional objectives are to support various display sizes, to support both keyboard and mouse input, to provide possibilities for application launching and to provide options for end-user customisation.

This chapter describes the design objectives in more detail (the *why*), and how SCOTZ achieves these objectives (the *how*). Chapter 7 will examine the two key features of SCOTZ (spatial stability and size morphing) in more detail, and provide an evaluation of SCOTZ and two common window switching interfaces.

6.1 Contributions and findings

- a description of the design objectives of SCOTZ and their theoretical foundations;

- a description of SCOTZ, including how it achieves the design objectives.

6.2 Design objectives

In this section, the two key design objectives (stability of the layout and support for revisitation) and the four additional objectives (support for various display sizes, support for keyboard and mouse input, support for application launching and options for end-user customisation) of SCOTZ are explained, including how related work provides support for the validity and usefulness of these objectives.

6.2.1 Stable layout

Spatial location memory is a person's memory of where objects are in space. It is well developed and can be extremely fast: studies have shown that items can be found in time proportional to the logarithm of the size of the set in stable layouts, which is much faster than the linear search time needed for unorganised sets (Cockburn et al., 2007). This fast performance is enabled by spatial stability, that is, items remaining in the same location over time. This idea has been known since the first interface design guidelines (Hansen et al., 1984). Many studies have demonstrated superior performance for spatially stable interfaces in comparison to a wide variety of alternatives (e.g., Teitelbaum and Granda, 1983). Also, several studies have found that users are able to learn the locations of a large number

of items in a user interface when the locations do not change (Czerwinski et al., 1999; Cockburn et al., 2007). Applying the idea of spatial stability in a window switcher implies that windows and applications should not move in the switcher's display. Within a work session, this design approach has clear advantages: since revisitation is strong (see Chapter 5), people will quickly learn the locations of the most frequently used windows. Spatial stability offers similar advantages across work sessions.

6.2.2 Support for revisitation

Chapter 5 identified strong window revisitation patterns, both *temporal* (users often switch back to previously used windows) and *frequential* (some windows are revisited more often than others). Explicit facilitation of this revisitation could lead to efficiency gains. Revisitation patterns have been identified in other areas of computer use as well, such as revisitation of certain regions within documents (Alexander and Cockburn, 2008), web site visits (Tauscher and Greenberg, 1997), command use (Greenberg and Witten, 1993) and menu use (Cockburn et al., 2007). Interfaces to support this revisitation have proven very successful (e.g., Alexander et al., 2009). Also, how to best support revisitation to web pages has been extensively studied (e.g., Kellar et al., 2006; Obendorf et al., 2007; Milic-Frayling et al., 2003; Cockburn et al., 2002, 2003; Kaasten and Greenberg, 2001). In general, systems to support revisitation usually support revisitation to recently used items (e.g., Alexander et al., 2009), frequently used items (e.g., Cockburn et al., 2007), or a combination of both (e.g., Kaasten and Greenberg, 2001). The results of Chapter 5 indicate that support for revisitation to both frequently and recently used windows can be beneficial.

6.2.3 Support for various display sizes

Modern display sizes range from very small (netbooks or tablet PCs) to very large and/or multi-monitor setups. Even 'everyday' setups can differ greatly in terms of the amount of screen estate (see Chapter 5). A window switching interface should ideally be suited for many different display sizes. On small screens navigational and functional controls, such as controls for switching between tasks can take up precious space. This leads to a difficult balancing act between allocating enough

space for the content to be displayed, while allowing enough space for the controls needed to interact with this content (Kamba et al., 1996). One proposed solution to this problem is the use of semi-transparent widgets (Kamba et al., 1996; Kärkkäinen and Laarni, 2002) and avoiding constantly displaying widgets that are only used occasionally as they take up too much space (Kärkkäinen and Laarni, 2002). On large displays, accessing items across large distances is difficult and time-consuming (Robertson et al., 2005; Czerwinski et al., 2006). Such 'distal access' could be avoided by, for example, placing controls in close proximity to the user's current mouse cursor position. As a design guideline, it is desirable that window switching tools support a wide range of display sizes.

6.2.4 Support for keyboard and mouse input

The log study in Chapter 5 showed that most users relied on mouse input for window switching, but that some users prefer keyboard-based methods, such as Alt+Tab. Apart from an inherent personal preference for a certain input device, convenience could influence the choice of input device (e.g., using the mouse when a coffee cup is held in the other hand, or using Alt+Tab when typing to avoid repositioning the hands). Also, the suitability of various pointing devices has been found to depend on the type of task (MacKenzie et al., 1991) and whether the input device is used with the preferred or non-preferred hand (Kabbash et al., 1993). As a design guideline, it is desirable that next generation window switching tools continue to support flexibility in input devices.

6.2.5 Support for application launching

In current interfaces, application launch facilities are largely partitioned from window switching tools (with the Microsoft Windows 7 Taskbar being a notable exception as applications can be 'pinned' to the Taskbar), yet these user activities are closely related. For example, to perform a search on the Internet the user needs to acquire a browser window. If the user starts by searching for a browser window they will be unsuccessful if no browser windows are active, necessitating a second action, i.e.,launching the browser. Alternatively, if the user begins by launching the application, they will often gain a superfluous window (when others were already available) adding to their window management load. By providing a single

interface mechanism for application launching and window switching the user no longer needs to consciously consider what to do (launch the application or switch to the window?) in order to be able to fulfill their current goal.

6.2.6 Options for end-user customisation

It is impossible to design a system that suits all users in all situations (MacLean et al., 1990), but providing options for end-user customisation can alleviate this problem, as users can select settings that suit them for their specific task. End-user customisation is an umbrella term for adaptations the user can make to a software application, such as how the application looks, behaves and how to interact with it, without the need to write code, with changes that are persistent across sessions (Mackay, 1991). End-user customisation is driven by a variety of motivations (Blom, 2000), such as the personal goals of the user, accommodation for individual differences, or personal preference. Options for end-user customisation can reduce interface complexity (Bunt et al., 2007; Stuerzlinger et al., 2006), because users can select a subset of functionalities that is most useful for them. In terms of purely aesthetic customisations, users have the need to personalise their computing space, and personalisation is important for the overall user experience (Tractinsky and Zmiri, 2005). Summarising, options for end-user customisation are important for the acceptance of a system, its usability and the overall user experiences and should therefore be available in new window switching interfaces.

6.3 SCOTZ

This section describes the new window switcher SCOTZ[1] (see Figure 6.1) and how it achieves the various design objectives identified in the previous section.

SCOTZ (Spatially Consistent Thumbnail Zones) groups windows by application in spatially stable *application zones*. Such semantically coherent and spatially grouped layouts have been shown to lead to very good performance in recalling item locations (Niemela and Saariluoma, 2003), and layouts that use grouping by semantic similarity lead to lower search times than ungrouped/random lay-

[1] SCOTZ is available for download at `http://www.cosc.canterbury.ac.nz/scotz`

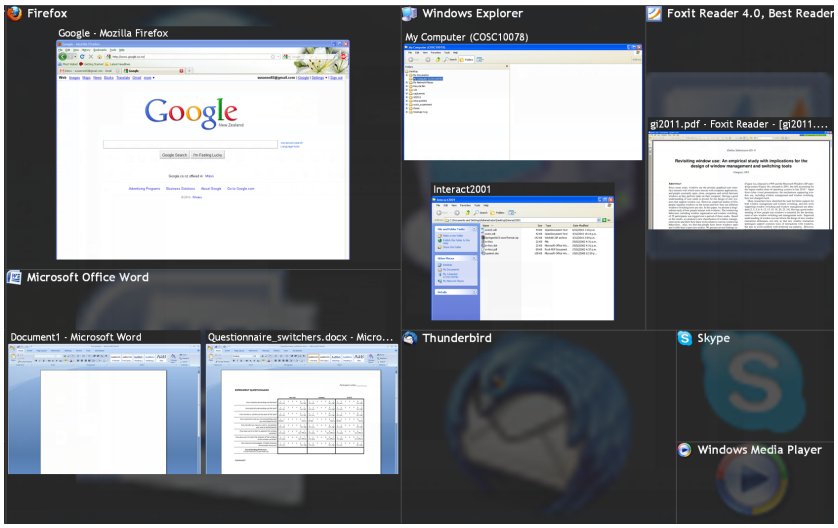


Figure 6.1: SCOTZ using the squarified treemap algorithm, showing seven applications and six windows.

outs (Halverson and Hornof, 2008). Also, as previous work has found that people can find items in a labelled hierarchy faster than in an unlabelled one (Hornof, 2001), SCOTZ shows the application icon and name in each application zone. The application zones' size and position are retained when the computer restarts, thereby providing a layout that is stable within sessions as well as between sessions. Spatial stability is applied in a hierarchical fashion: applications are given stable zones in the switcher display, since these change slowly over the long term; and within each zone, the application windows used in the current work session are placed in stable spatial locations. Also, the application-based organisation provides an initial guide to the location of a new window as it quickly narrows the users search space: rather than having to search the whole display, users can quickly dismiss the majority of candidates by homing in on the appropriate application zone.

Additionally, size morphing is applied to these application zones. Size morphing enables frequent targets to be enlarged to enhance their visibility and to reduce pointing time. Also, using a morphing layout allows the introduction of new items. The size morphing implemented in SCOTZ is (abstractly) shown in Figure 6.2; the more often an application is switched to, the bigger the size of the application zone becomes. Another example is shown in Figure 6.3. Applica-

tion zone sizes are an *exact* reflection of how many times these applications are switched to; if application 'A' is switched to twice as often as application 'B', application zones 'A' is twice as big (in terms of surface area) as application zone 'B'. Figure 6.1 shows an actual SCOTZ window in full-screen mode with seven application zones and six windows.

The following sections describe in more detail how SCOTZ achieves the design objectives presented in the previous section.

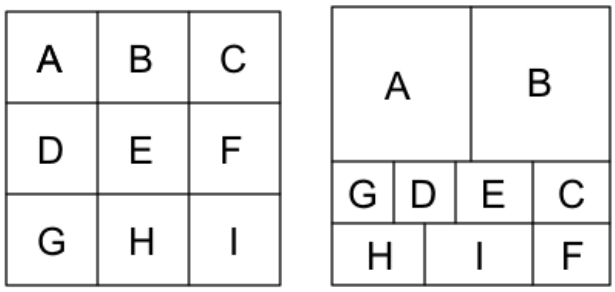


Figure 6.2: Abstraction of the design of SCOTZ with nine application zones, labelled A to I: (a) before size morphing, (b) after several iterations of size morphing; application A and B have been switched to frequently, and are therefore allocated more space, while keeping all application zones in relatively stable positions.

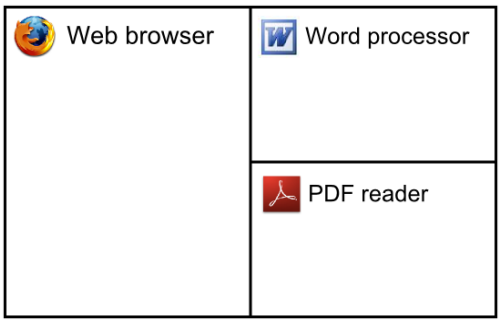
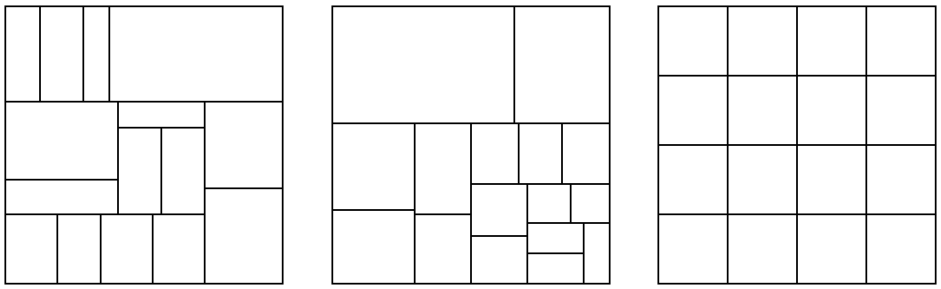


Figure 6.3: An example of size-morphed application zones; the web browser is switched to twice as often as the word processor and the PDF reader, and is therefore allocated twice as much space.

6.3.1 *Stable layout*

SCOTZ groups windows by application in application zones, which stay in as stable locations as possible. SCOTZ offers three different layouts for the application zones: two treemap layouts, where size morphing is applied, and a grid layout where no size morphing is applied (see Figure 6.4). A treemap is a space-filling layout that recursively divides the screen in rectangles, sized relative to some underlying data attribute (Johnson and Shneiderman, 1991). In SCOTZ, the size of the application zone reflects how often an application has been switched to. Various algorithms for generating treemaps exist, each offering advantages for specific contexts. For example, some treemaps are designed to keep items in relatively stable positions as the underlying data changes. The properties of treemaps and the suitability of various treemaps to be used for the layout of the application zones in SCOTZ are examined in more detail in Chapter 8.



(a) Spiral treemap (b) Squarified treemap (c) Grid layout

Figure 6.4: Three layout options for the application zones in SCOTZ.

The ordering of the window representations within each application zone is either alphabetically by window title, by frecency (a combination of frequency and recency), or the same as the order in which the respective windows were opened, using a row-major order. Alphabetic ordering is very predictable and understandable, sorting by frecency supports window revisitation and sorting the representations by the order in which the respective windows were opened is very stable when additional windows are opened for an application.

6.3.2 *Support for revisitation*

SCOTZ allocates more space to the most *frequently* switched to applications when a treemap layout (see Figure 6.4) is used. This size morphing reduces the Fitts' Law (Fitts, 1954) targetting time of these application zones, as well as making them easier to find because the size difference of items facilitates guided search (Wolfe and Horowitz, 2004).

SCOTZ supports switching back to *recently* used windows by retaining the recency-like z-ordering that is used in Windows Alt+Tab. Alt+Tab is very powerful when the user needs to 'flip' between a few recent windows. When SCOTZ is bound to Alt+Tab, pressing Alt+Tab will bring up the SCOTZ interface as normal, but repeated presses of Tab will cycle through the windows according to z-order, similar to the implementation of Microsoft Windows Alt+Tab (see Figure 6.5). This provides quick and easy access to recently used windows.

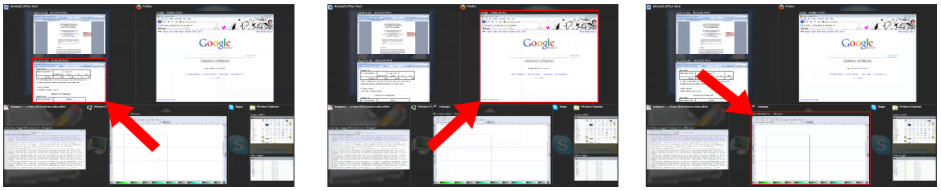


Figure 6.5: Using Alt+Tab to cycle through the windows in SCOTZ by z-order. A pulsating red border indicates the currently selected window (indicated by a red arrow in this figure).

6.3.3 *Support for various display sizes*

To accommodate for a wide range of display sizes SCOTZ can either be a full screen window (see Figure 6.1), a smaller window in the centre of the screen, or a smaller window positioned under the mouse cursor (see Figure 6.6). By positioning SCOTZ under the mouse cursor it provides proximal access on (very) large displays.

Also, as SCOTZ is a transient interface, which is only shown when the user intentionally invokes it, it does not take up precious screen estate when it is not used, which is particularly important on small screens.

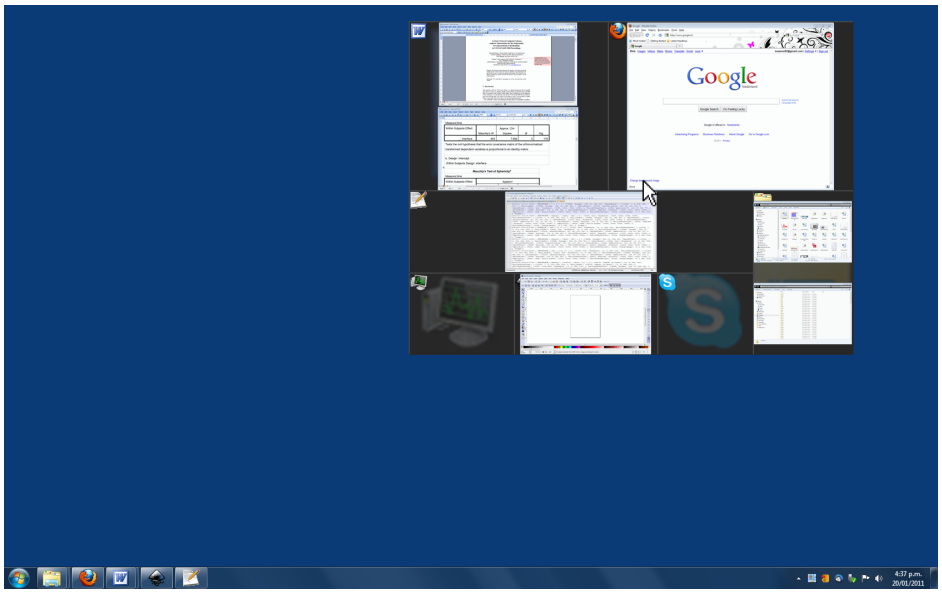


Figure 6.6: SCOTZ in a smaller display mode, positioned under the mouse cursor.

6.3.4 Support for keyboard and mouse input

SCOTZ has rich options for both keyboard and mouse input. Both the mouse and the keyboard can be used to invoke SCOTZ; the user can bind any key (combination), middle mouse button click or left+right mouse button click to SCOTZ. If SCOTZ is bound to the Alt+Tab key combination it retains the z-order-based window switching as it is implemented in Windows Alt+Tab.

As the design of SCOTZ is aimed at keeping the application zones in spatially stable locations, positioning the SCOTZ interface relative to the mouse cursor means that a target can always be acquired with the same mouse gesture. Previous work has shown that despite some difficulties in the learning/training phase, mouse gestures can be very efficient and accurate (Dulberg et al., 1999).

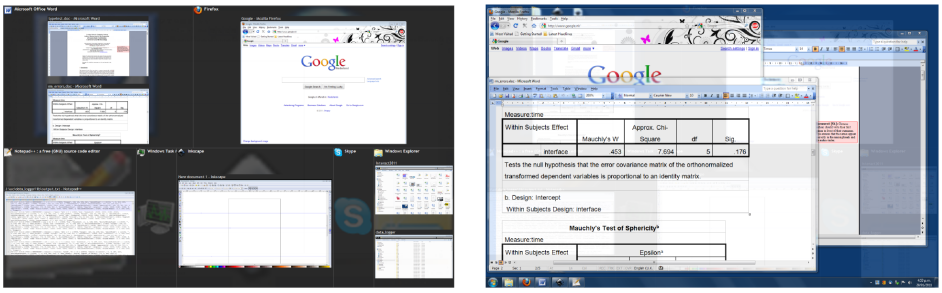
6.3.5 Support for application launching

SCOTZ provides a single interface mechanism for both window switching and application launching. After having used SCOTZ for a long period of time, users will most likely be familiar with the locations of application zones (as these stay in spatially stable locations). Therefore, it makes sense for these zones to double as efficient application launchers. If an application has no open windows associated

with it, the application zone is still visible in SCOTZ (for example, see the Skype, Thunderbird and Window Media Player zones in Figure 6.1) and clicking on an empty application zone launches the application.

6.3.6 Options for end-user customisation

Users can make several functional personalisations in SCOTZ to accommodate their personal goals and preferences (in addition to choices about layout and input methods, as presented in previous sections): for example, users can include or exclude certain applications in SCOTZ, and can customise the rate of growth for application zones. The appearance of SCOTZ is also customisable: users can change the font size and type to accommodate various levels of visual acuity, and the colour scheme to accommodate for various types of colour blindness as well as personal taste and preference. Also, users can adjust the opacity of SCOTZ (see Figure 6.7).



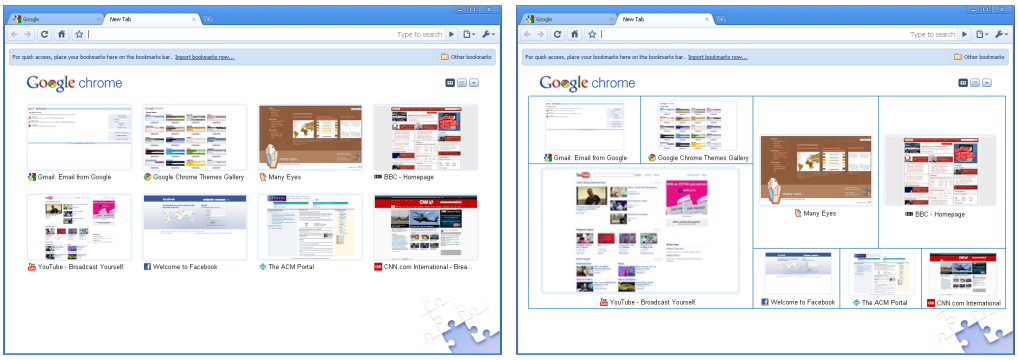
(a) Opaque (b) Almost transparant

Figure 6.7: Different opacity levels of the SCOTZ window.

6.4 Conclusion

This chapter presented SCOTZ, a new window switcher which is designed to provide a spatially stable layout and to provide support for window revisitation. The design objectives presented in this chapter could be applied to other areas of HCI as well.

An example of how the two key design objectives (stability of the layout and support for revisitation) could be applied in other areas of human-computer in-



(a) (b)

Figure 6.8: (a) Google Chrome's New Tab functionality, (b) a mock-up of Google Chrome's New Tab functionality using a treemap.

teraction is shown in Figure 6.8. The web browser Google Chrome shows the websites you visit most often whenever you open a new tab (see Figure 6.8a). As noted in section 6.2.2, web site visits are known to follow a very skewed revisitation pattern: a few web sites are visited a lot, but the relative frequencies quickly drop off (Tauscher and Greenberg, 1997). By making the size of the website thumbnail reflect the visit frequency to that particular website the Fitts' Law targetting time for frequently occurring targets is reduced. A mock-up of Google Chrome's New Tab layout using a treemap is shown in Figure 6.8b.

In general, providing a spatially stable layout, supporting various display sizes, supporting keyboard and mouse input and providing options for end-user customisation are useful objectives to guide the design of many other user interfaces.

Chapter VII

SCOTZ: Theoretical and Empirical Validation

Chapter 6 described SCOTZ and its design objectives. This chapter presents theoretical examinations and various empirical studies which assess the validity of the following underlying assumptions of SCOTZ: (1) providing a stable layout leads to efficiency gains and (2) size morphing leads to improved item targetting times and makes it easier to find items. Also, this chapter presents the findings from a qualitative study, and an empirical study which demonstrates that SCOTZ yields performance and preference benefits over existing window switching tools.

The first key design principle of SCOTZ is to provide a spatially stable layout. Previous work has shown the performance benefits of spatially stable layouts over a variety of alternatives (e.g., Teitelbaum and Granda, 1983). To examine the effect of spatial instability in more detail, this chapter first presents two experiments: (1) an empirical study which compares spatially stable 2D layouts with frequency and recency based layouts, which are two orderings that could be used in a window switcher, and (2) an empirical study which examines whether the (slight) instability caused by size morphing negatively affects user performance.

The second key design principle of SCOTZ is to use size morphing, i.e., allocate more screen estate to the most frequently switched to applications. This size morphing has two potential benefits: (1) improved item targetting times, as encapsulated by *Fitts' Law*, and (2) improved item search times, because the size difference of items facilitates *guided search*. Therefore, the second section of this chapter provides theoretical proof of the improved item targetting time of items in size-morphed layouts, and the third section provides an empirical and an eye-tracker study that demonstrate size-morphed layouts facilitate finding items.

The final two sections of this chapter present a qualitative evaluation of SCOTZ and an empirical evaluation of SCOTZ and two other state-of-the-art window switching interfaces: the Microsoft Windows 7 Taskbar and Windows 7 Alt+Tab.

7.1 Contributions and findings

- providing a spatially stable layout is important for user performance;
- the slight spatial instability caused by size morphing does not have a large negative effect on user performance;
- size-morphed layouts, including treemap layouts, have efficiency benefits;
- size-morphed layouts assist users in finding items;
- lessons learned from a qualitative study of SCOTZ;
- an empirical study demonstrating the performance benefits of SCOTZ over two major commercial window switching interfaces (the Windows 7 Taskbar and Windows 7 Alt+Tab).

7.2 Spatial stability

The first experiment[1] described in this section examines the importance of spatial stability in a layout, or whether orderings such as by frequency or recency lead to equally good performance as fully stable layouts. The second experiment described in this section is inspired by the observation that items unavoidably move as they grow/shrink when a size-morphed layout is used, which affects the stability of the layout as a whole. Therefore, the experiment examines how severely, if at all, this instability affects user performance.

7.2.1 Experiment: Stable, frequency, or recency?

As described in Chapter 6, previous work suggests that spatially stable layouts lead to better performance than spatially unstable layouts. However, such a stable ordering is an unattainable 'gold standard' for SCOTZ, as total stability would prohibit the addition of zones for new windows/applications. However, previous studies commonly only compare stable layouts to randomly updating layouts (e.g.,

[1] All experiments described in this chapter were conducted at different times with different groups of participants.

Teitelbaum and Granda, 1983). A window switching interface can order items in a variety of ways, including by recency (similar to Alt+Tab) or by frequency (which might work well for Zipf-like distributions). The goal of the experiment described in this section to examine user performance on a fully stable layout (though unattainable for practical use in SCOTZ, it is an informative control condition) to other 'sensible', but less stable orderings: recency and frequency. This is achieved by studying the performance impact of several different orderings (stable, recency order, frequency order, and random order) for tasks involving acquisition of targets in a Zipf-like distribution (similar to the distribution found for window revisitation in Chapter 5).

Layouts

Four layouts were used: *stable*, *recency*, *frequency* and *random* (a useful control condition). The *stable* layout used an arbitrary but stable order (i.e., icons never moved). The *recency* layout moved the most recently selected item to the top left of the grid, pushing earlier items along in row-major order (similar to Alt+Tab). The *frequency* layout repositioned items according to their cumulative selection counts (most frequent at top left, in row-major order). Finally, in the *random* layout all windows were randomly repositioned after each selection.

Procedure

The experimental interface consisted of a grid of distinct icons (see Figure 7.1) with a cued target on the right. Participants were instructed to click on the target icon region as quickly and accurately as possible. After each successful selection the windows were temporarily hidden from view, participants had to press a 'next' button in the centre of the layout, and the layout was updated. To understand how performance with these interfaces is influenced by number of targets, participants completed trials with 4, 9, 16, 25, 36, 49 and 64-item grids. The Zipf's Law distribution of targets was generated by randomly selecting eight targets from among the candidates (all four for the 4 item grid): one was cued 10 times, one 5 times, then 3, 2, 2, 1, 1, and 1 for the others (power law $R^2 \approx .97$, $\alpha \approx 1.3$).

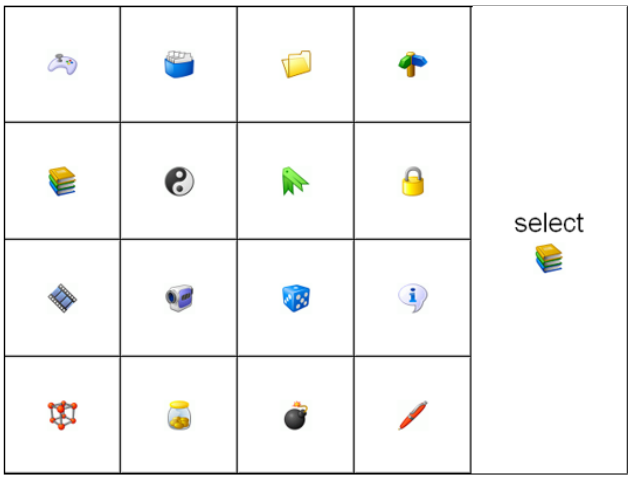


Figure 7.1: Experimental interface showing 16 icon regions and the target on the right.

Design

The *selection time* dependent measure was analysed using a 4×3×7 RM-ANOVA for factors *layout* (stable, frequency, recency, random), *experience* (low, medium or high) and *items* (4-64). Experience was determined by assigning first-time icon selections as low experience, 2^{nd}-7^{th} selections as medium experience, and 8^{th}-10^{th} as high experience.

The experiment used a within-subject design, with all participants performing tasks in all layouts. The different layouts were presented in random order, and within that order the different set sizes were presented randomly. Icons that had been selected more than once in a layout were not re-used in following conditions. The interface was displayed on a monitor with 1280 × 1024 pixels resolution.

Participants

Twenty-six students volunteered for the experiment (16 male, 10 female, 16-44 years old). Their participation lasted approximately 45 minutes.

Results and discussion

All correct selections preceded by an erroneous click on a wrong item were removed. The overall error rate was 3.1%, but there was no difference across con-

ditions in terms of errors made. All extreme outliers (data points larger than 3 standard deviations above the mean) for each *layout*×*experience*×*items* condition were removed.

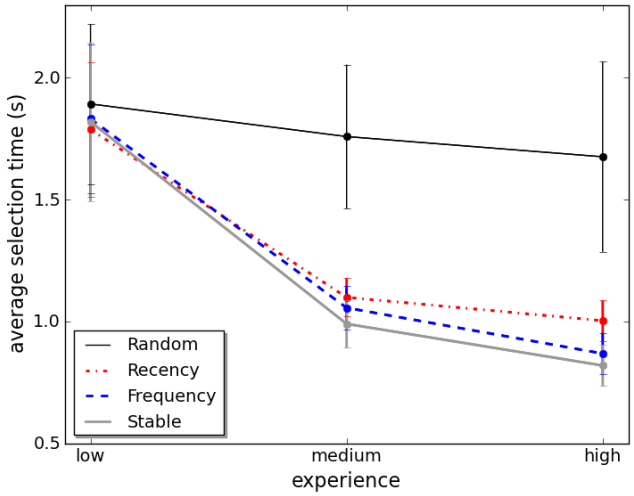


Figure 7.2: Experiment mean selection times for the four *layouts* by *experience* including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

All factors showed significant main effects[2]: *layout* ($F_{1.4,27.9}$=72, p<.001), *experience* ($F_{2,40}$=409, p<.001) and *items* ($F_{2.6,51.1}$=135, p<.001). Stable layouts were the fastest (mean 1.2s) followed by frequency ordering (1.3s), recency ordering (1.3s), and random (1.8s). Post hoc comparisons (Bonferroni correction, α=.05) show pairwise differences between all layouts and the random layout, and between the stable layout and the recency layout. Figure 7.2 shows a significant *layout*×*experience* interaction ($F_{3.0,59.0}$=47, p<.001) caused by relatively constant performance across *experience* with the random layout in contrast to marked improvement with other layouts. Figure 7.3 shows a significant *layout*×*items* interaction ($F_{4.4,8.3}$=14, p<.001), caused by the random layout worsening much more rapidly across increased number of items than the other three layouts.

[2] Analysis corrected for deviance from sphericity by means of Greenhouse-Geisser where required.

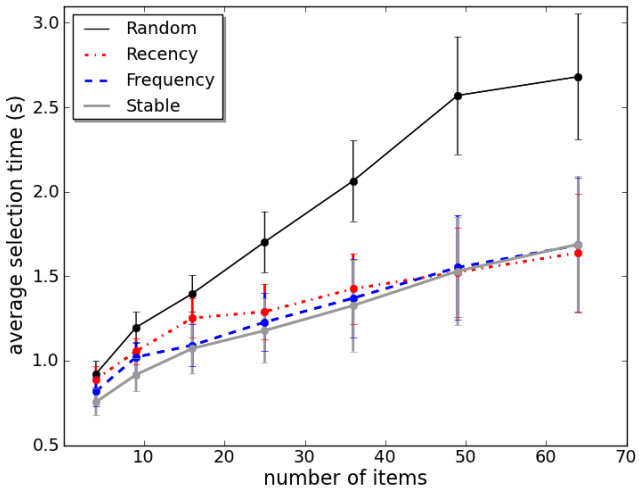


Figure 7.3: Experiment mean selection times for the four *layouts* by *items* including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

The results show that stability is beneficial: the stable, frequency, and recency layouts all supported expertise development, while random did not.

Performance with stable and frequency layouts was similar to each other, raising the question of whether SCOTZ should use frequency ordering. However, the experiment used a Zipfian distribution of stimuli, causing the frequency layout to quickly settle to spatial stability, so the frequency layout's comparative success is probably also explained by its spatial stability (after a short period of reorganisation). In practical use, a frequency layout for windows is unlikely to succeed due to the transient nature of most windows. Consequently, some other basis for organisation is required, such as application zones. Although application zones could be placed in a frequency layout, doing so would create placement instability during early use, which might cause users to discard the system due to its unpredictable behaviour. Therefore, using spatial stability as the main layout principle is a preferable solution.

The finding that the stable layout was significantly faster than the recency layout, with performance benefits increasing with expertise is easily explained: after each selection in the recency layout users must either calculate the new position of their targets or visually search for them, both of which demand time.

7.2.2 *Experiment: Does size morphing cause too much spatial instability?*

The previous experiment demonstrated that total stability is the 'gold standard', but total stability would prohibit the addition of zones for new windows/applications, as well as prohibiting morphing sizes to reduce the Fitts' Law targetting time. Size-morphed layouts do allow for the addition of new zones, and lead to Fitts' Law targetting time benefits (see section 7.3). Unfortunately, size morphing inevitably introduces some instability as items grow and shrink and 'push' others out of the way.

The experiment described in this section examines the performance impact of spatial instability caused by the morphing behaviour by examining the performance of a totally stable layout (the gold standard control condition) and two layouts that use size morphing, and are therefore inherently unstable. Similar to the experiment described in the previous section, the performance impact of several different layouts (stable, squarified treemap and spiral treemap) for tasks involving acquisition of targets in a Zipf-like distribution was studied. Both treemaps used in the experiment are relatively stable: the squarified treemap should rapidly stabilise because of Zipfian target distribution, while the spiral treemap is, by design, quite stable in terms of item placement. For more on treemap and their properties, see Chapter 8.

Layouts

Three layouts were used: *stable*, *squarified treemap* and *spiral treemap*. The *stable* layout used an arbitrary but stable order (i.e., icons never moved), where *no* size morphing was applied. Morphing behaviour was implemented with squarified (Bruls et al., 2000) and spiral (Tu and Shen, 2007) treemaps. These treemaps were used in this experiment because they are both good candidates for SCOTZ because of their favourable individual properties (see Chapter 8).

Procedure

The procedure was similar to the experiment in Section 7.2.1. In the treemap conditions the item sizes were related to how often the items had been selected during the course of the experiment, with each item selection causing that item to be allocated more space. The treemap layouts started off as a grid layout where

items were all of equal size, and item sizes were updated after each successful selection (for an example, see Figure 7.4). To isolate the effect of stability, a very conservative 'rate of growth' of the items was used (similar to the *damping* effect in Cockburn et al., 2007), such that there was no Fitts' Law targetting time advantage for the layouts that used size morphing compared to the stable (grid-like) control condition.

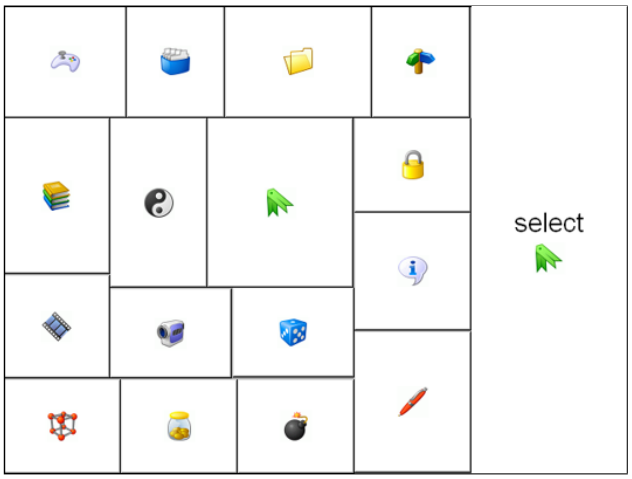


Figure 7.4: Experimental interface showing 16 icon regions and the target on the right. The items are layed out using the spiral treemap algorithm and item sizes reflect how often they have been selected.

Design

The design was similar to the experiment in Section 7.2.1. The *selection time* dependent measure was analysed using a 3×3×7 RM-ANOVA for factors *layout* (stable, squarified, spiral), *experience* (low, medium or high), and *items* (4-64). Experience was determined by assigning first-time icon selections as low experience, 2^{nd}-7^{th} selections as medium experience, and 8^{th}-10^{th} as high experience.

Participants

Seventeen students volunteered for the experiment (15 male, 2 female, 21-35 years old). Their participation lasted approximately 30 minutes.

Results and discussion

The overall error rate was 1.7%, but there was no difference across conditions in terms of errors made. All extreme outliers (data points larger than 3 standard deviations above the mean) for each *layout*×*experience*×*items* condition were removed.

There were significant main effects[3] for *items* ($F_{2.6,41.7}$=204, p<.001) and *experience* ($F_{1.1,16.8}$=444, p<.001), but not for *layout* ($F_{2,32}$=1, p>.3), nor were there any interactions. Mean item selection times for the three layouts were similar at 1.30s, 1.34s and 1.35s for stable, squarified and spiral respectively (also see Figure 7.5).

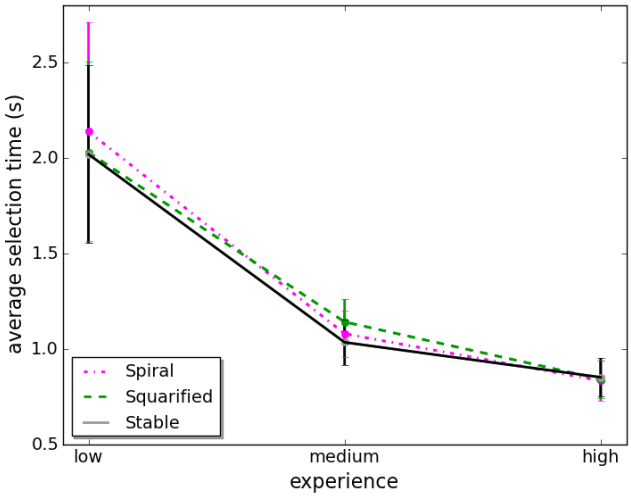


Figure 7.5: Experiment mean selection times for the three *layouts* by *experience* including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

The results shows that the spiral and squarified treemaps were only marginally slower than the (non-morphed) stable control. This means that the morphing effect of the two treemaps and the instability that unavoidably comes with it, did not substantially adversely affect performance.

[3] Analysis corrected for deviance from sphericity by means of Greenhouse-Geisser where required.

7.3 Size morphing and targetting time - theoretical analysis

This section examines whether size morphing leads to improved item targetting times. This is achieved by calculating the item targetting times in size-morphed layouts as encapsulated by Fitts' Law.

Fitts' Law (Fitts, 1954) is a well-known and thoroughly tested rule of target acquisition time. Fitts' Law states that the time required to move to a target (movement time, *MT*) is a function of both the width of the target (*W*) and distance to the target (amplitude, *A*). More precisely, the following relationship has been proposed and validated (MacKenzie and Buxton, 1992), where *a* and *b* are empirically derived constants:

$$MT = a + b \times log_2(\frac{A}{W} + 1) \tag{7.1}$$

The part $log_2(\frac{A}{W} + 1)$ in Equation 7.1 is referred to as the *index of difficulty* (*ID*). For rectangular items, the shortest side can be used as the target width *W* (MacKenzie and Buxton, 1992).

By using size morphing, more frequently targetted items become larger and therefore their Fitts' Law targetting time is decreased. However, the Fitts' Law targetting time of items that are not (often) selected increases as these items become smaller. This 'trade-off' most likely leads to a lower average Fitts' Law targetting time for this very same reason: the larger items are selected more often, so their improved Fitts' Law targetting times *contribute more* to the average targetting time of the layout than the worsened Fitts' Law targetting time of the smaller items.

To examine this potential beneficial effect of size morphing (improved targetting times) the following section calculates the Fitts' Law targetting time of size-morphed layouts under 'ideal' circumstances (i.e., all items are perfectly square). This is followed by a calculation of the Fitts' Law targetting time of *treemap* layouts.

7.3.1 Can size morphing lead to improved item targetting times?

This section provides a proof of concept of the advantage of using size morphing by demonstrating that the weighted average index of difficulty (*ID*) of a layout

that uses size morphing layout is lower than that of a layout that does not use size morphing, under the assumption that all items are perfectly square. Also, item sizes are an exact representation of their underlying value; an item with value '10' is ten times larger (in terms of surface area) than an item with value '1'. Using *weighted* average then implies that the *ID* of an item with value '10' contributes 10 times more to the average *ID* than the *ID* of an item with value '1', as the former is selected 10 times as often as the latter, see Equation 7.2.

$$\overline{ID(weighted)} = \frac{\sum_{i=1}^{n} value_i \times ID_i}{\sum_{i=1}^{n} value_i} \tag{7.2}$$

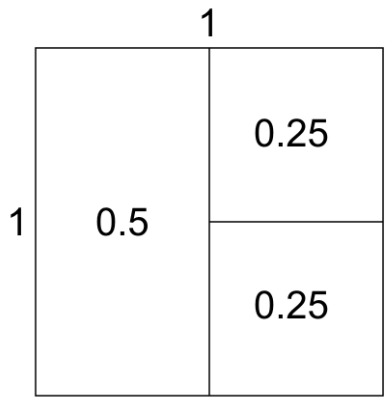


Figure 7.6: A unit square containing three items. Inset are the respective values/-sizes of the items.

The proof of concept uses a unit square (see Figure 7.6), but the proof applies equally well to larger layouts as the Fitts' Law is scale-independent (as it uses the ratio of distance to width, which is constant regardless of scaling). For ease of calculation, all item values add up to one.[4] Because of this conversion, the sizes of the items (expressed as their surface areas) are equal to their values (see Figure 7.6). The calculation assumes perfectly square items, which makes the *W* (width) of the item the square root of its value (as the item size is equal to the value of the item):

[4] Any data set can be converted to a set where all values add up to one by converting the absolute data values to (relative) ratios or percentages.

$$ID_i = log_2(\frac{A}{\sqrt{value_i}} + 1) \tag{7.3}$$

The weighted ID for each item is therefore:

$$ID(weighted)_i = value_i \times log_2(\frac{A}{\sqrt{value_i}} + 1) \tag{7.4}$$

By randomly picking a point in a unit square the expected distance to the centre of the square is approximately .382598 (Finch, 2003). By inserting this as the value for *A* in Equation 7.3 the weighted index of difficulty becomes a function of item value (see Figure 7.7).

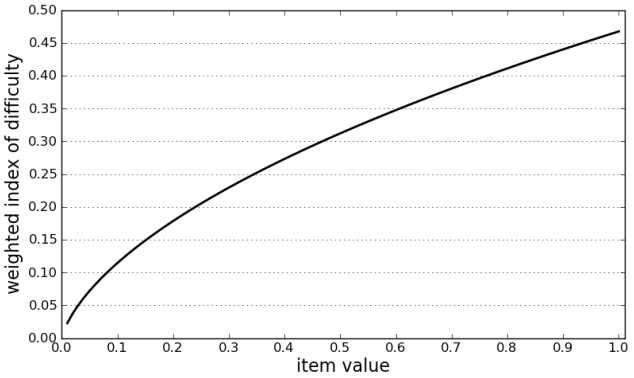


Figure 7.7: Weighted index of difficulty by item value, for items in a unit square.

Inserting Equation 7.4 in Equation 7.2 gives Equation 7.5 for the weighted average *ID* (the denominator of Equation 7.2 disappears as all item values to add up to one because a unit square is used).

$$\overline{ID(weighted)} = \sum_{i=1}^{n} value_i \times log_2(\frac{A}{\sqrt{value_i}} + 1) \tag{7.5}$$

A basic example of how size morphing benefits overall Fitts' Law targetting time is shown in Figure 7.8. In this example, there are only two items. If both items are assigned 50% of the available area, the weighted average *ID* (from Figure 7.7 and Equation 7.2) is 2 $\times$ 0.31 = 0.62, as indicated in Figure 7.8. If one items is assigned 80% of available space (because it is selected 80% of the time), and the other one, therefore, 20% of available space, the weighted average *ID*

decreases by Δx and increases by Δy, as indicated in Figure 7.8, making the weighted average *ID* 0.18 + 0.41 = 0.59.

Because of the nature of the data distribution $\Delta x > \Delta y$, therefore, the weighted average *ID* always decreases when size morphing is used compared to a layout where all items are of equal size.

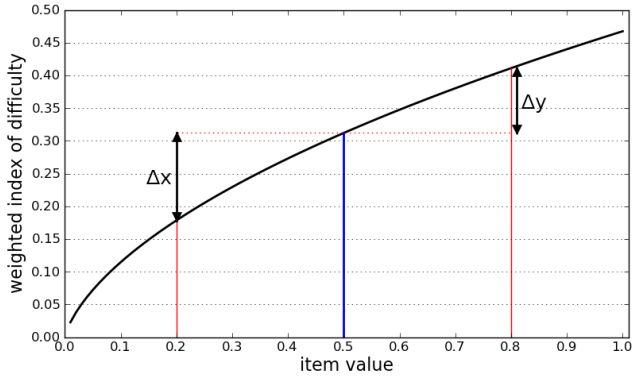


Figure 7.8: Example of the Fitts' Law advantage of using size morphing (see explanation in text).

The weighted average *ID*'s for several data distributions are shown in Figure 7.9. For example, in the most pronounced distribution, when the item values follow a power distribution of the form $y \sim \frac{c}{x^2}$, the most selected item has a value of approximately 0.62. This means it is the selection target for 62% of selections and therefore takes up 62% of the total area of the layout. Figure 7.9 shows the weighted average index of difficulty for several distributions for various numbers of items, as well as the weighted average index of difficulty of a layout that does not use size morphing. The results show that (1) size morphing can lead to lower item targetting times and (2) the more pronounced (or skewed) the data distribution, the more beneficial size morphing becomes.

7.3.2 Can treemap layouts lead to improved item targetting times?

The calculations in the previous section assume that all items in the layout are perfectly square. However, this is unattainable for *actual* treemap layouts. While many treemap algorithms are aimed at creating a layout with as square *as possible* items, they are never all perfectly square due the constraints of a treemap layout,

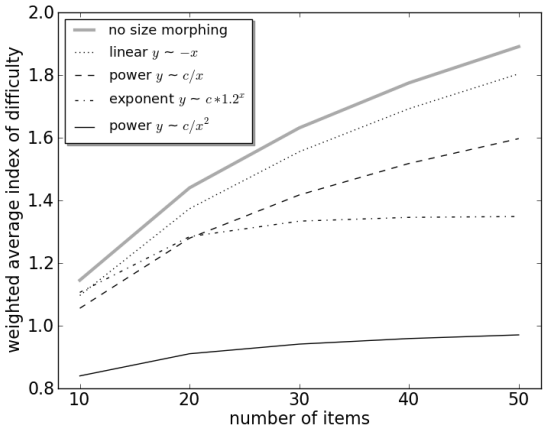


Figure 7.9: Weighted average index of difficulty for size-morphed layouts for the various data distributions, for various numbers of items. Also shown is the weighted average index of difficulty when no size morphing is used (and all items are of equal size).

i.e., item sizes must reflect underlying data values and the layout must be space-filling while items may not overlap (for more on treemaps and their properties, see Chapter 8). Therefore, the weighted average ID of a treemap layout will never reach the *optimal* values shown in Figure 7.9.

Figure 7.10 shows the weighted average ID for squarified (Bruls et al., 2000), spiral (Tu and Shen, 2007) and Moore (see Chapter 8) treemaps for various numbers of items when the data follows a power distribution of the form $y \sim \frac{c}{x}$, calculated in the same manner as the results in the previous sections.

The results in Figure 7.10 show that not all treemaps layout improve item targetting times compared to a 2D layout where no size morphing is applied. Also, the results show that the squarified treemap leads to the relatively lowest item targetting times.

7.4 Size morphing and search time

Pointing to the item, as captured by item targetting time in the previous section, is only a part of total item acquisition time. When the user does not know the item location, he/she must also search for it. Theory suggests that size morphing assist users when searching for items, as the differences in item size makes finding par-

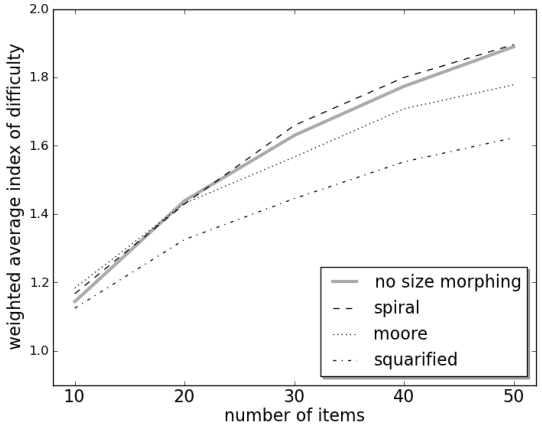


Figure 7.10: Weighted average ID for three treemap layouts for a power distribution of the form $y \sim \frac{c}{x}$, for various numbers of items. Also shown is the weighted average ID when no size morphing is used.

ticular items easier because of *guided search* (Wolfe et al., 1989; Wolfe and Gray, 2007). Guided search is an influential theory that offers an explanation of how people (visually) search for targets. According to the guided search theory people can focus their visual attention to a certain stimulus attribute, and exclude other input as noise, in the very early stages of visual processing. In other words, in the early stage of visual processing attention can already be guided to select appropriate objects in later stages of visual processing. Certain attributes of items are identified as 'guiding attributes'. Colour, motion and size are "undoubted" [*sic*] attributes that guide search (Wolfe and Horowitz, 2004, p. 500). The presence of one of these attributes can greatly facilitate finding items, as attention can be 'guided' to them in the early stages of visual processing.

In a size-morphed layout the most used items are relatively large, which should facilitate finding these items: if the participant has selected an item quite often (and understands that item size is directly mapped on item selection count), attention is guided toward the large items in the layout.

This section presents two studies: an empirical study similar to the studies described in section 7.2 and an eye-tracker study, which examine whether there is evidence that size-morphed layouts assist users in finding items.

7.4.1 Experiment: Does size morphing support guided search?

The goal of the experiment in this section is to examine whether there is evidence that size-morphed layouts lead to improved item search times. This is achieved by presenting participants a 'random treemap' layout. In this random treemap, size morphing is applied consistently and predictably, but the item locations are assigned randomly after each item selection. Theory (Wolfe and Horowitz, 2004) suggests that the size difference between items in such a layout should assist users in finding items. Therefore, it is reasonable to expect that this 'random treemap' will support expertise development, as opposed to a randomly updating layout where all items are of equal size (see Figure 7.2). Also, if the results indicate that random treemap supports expertise development, this can not be explained by people learning item locations, as they are completely unpredictable and instable. However, guided search can explain this result.

Layouts

The experiment compared target acquisition performance using the random treemap (described above) to the more stable spiral and squarified treemaps, as well the 'gold standard' fully stable layout that does not use size morphing.

Procedure

The procedure was similar to the experiments in sections 7.2.1 and 7.2.2. Participants completed trials for all four layouts (the random, spiral and squarified treemap, and the stable grid layout), which were presented in random order. However, contrary to the experiments presented in previous sections, where participants completed trials with grids with various sizes, only a 25-item grid was used in the current study to limit the total duration of the experiment. Also, participants made more selections in each layout compared to the previous experiments. This ensures an absence of expertise development can not be attributed to the experiment being too short. The Zipf's Law distribution of targets was generated by randomly selecting nineteen targets from among the candidates: one was cued 19 times, one 10 times, then 6, 5, 4, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1 and 1 for the others (power law $R^2 \approx .97$, $\alpha \approx 1.0$).

Design

The design was similar to the experiment in Sections 7.2.1 and 7.2.2. The *selection time* dependent measure was analysed using a 4×3 RM-ANOVA for factors *layout* (stable, squarified, spiral, random) and *experience* (low, medium, high). Experience was determined by assigning first-time icon selections as low experience, 2^{nd}-15^{th} selections as medium experience, and 16^{th}-19^{th} as high experience.

Participants

Thirty students volunteered for the experiment (17 male, 13 female, 21-35 years old). Their participation lasted approximately 30 minutes.

Results and discussion

The overall error rate was 1.7%, and there was no difference across conditions in terms of errors made. All extreme outliers (data points larger than 3 standard deviations above the mean) for each *layout*×*experience* condition were removed. The results are shown in Figure 7.11.

There were significant main effects[5] for *layout* ($F_{3,87}$=35, p<.001) and *experience* ($F_{1.4,40.6}$=562, p<.001). Mean item selection times were 1.3s, 1.4s, 1.4s and 1.6 for stable, squarified, spiral and random layouts, respectively. Post hoc comparisons (Bonferroni correction, α=.05) show pairwise differences between all layouts except the spiral and squarified layout. Figure 7.11 shows a significant *layout*×*experience* interaction ($F_{3.847,111.561}$=10, p<.001), caused by different performance of the various layouts on the *medium experience* level.

Overall, these results suggest that the random treemap assists users in finding items. This is particularly apparent when comparing the results in Figure 7.11 and Figure 7.2; a random layout that does not use size morphing (Figure 7.2) does not support expertise development, but a random layout that uses size morphing does support expertise development (Figure 7.11). These results can not be explained by participants learning item locations as it is not possible to learn item locations in a layout that randomly updates item positions. *Guided search*, however, can ex-

[5] Analysis corrected for deviance from sphericity by means of Greenhouse-Geisser where required.

plain these findings. Finding large items is facilitated by applying size morphing, regardless of the (in)stability of the layout.

Furthermore, the *layout*×*experience* interaction can also be explained by guided search: on the *medium experience* level the size difference between items is not very big (yet), hence guided search is not yet facilitated. On the *high experience* level, when the size difference between items is substantial, guided search is facilitated and this eliminates the negative impact of instability (note that on the *high experience* level the random layout performs comparable to the other layouts).

Finally, the *layout*×*experience* interaction also reveals that the spiral and squarified layouts perform worse than the stable layout on *medium experience* level (albeit better than the random layout), a result that was not found in the experiment described in Section 7.2.2. Apparently, the slight instability of the spiral and squarified layouts does impair performance, but the effect is weak and therefore hard to detect.

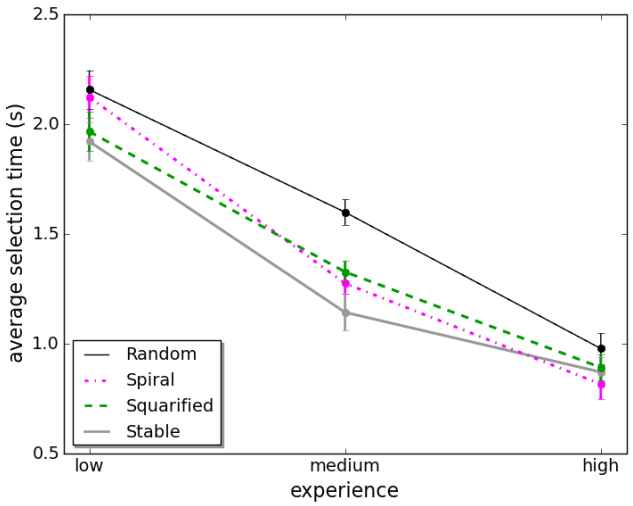


Figure 7.11: Experiment mean selection times for the three *layouts* by *experience* including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

7.4.2 Eye-tracker study

The experiment described in the previous section provides preliminary evidence that size-morphed layouts facilitate finding items. This section provides a more 'direct' method of assessing the validity of this hypothesis by examining whether it can be supported by eye-tracker data. Most likely, size-morphed layouts make it easier to find items, because attention is *guided* toward the target item. Eye focus, as recorded by an eye-tracking device, usually indicates where attention is directed at (Poole and Ball, 2005). Therefore, the use of an eye-tracker can shed more light on whether the size difference between items indeed makes it easier to find certain items in a layout. In other words, with the use of an eye-tracker the visual search component of item selection time (as measured in the previous experiment) is isolated.

Eye-trackers allow for unobtrusive recording of a participant's eye focus. Eye-trackers log how the eye gaze shifts between and fixates on items in a display. From this data, information about the visual search process can be extracted. For example, many shifts and hardly any long fixations can imply that a person is searching for an item (Poole and Ball, 2005). The following section describes the data generated by eye-trackers in more detail, as well as desirable settings for eye-trackers for studies similar to the current one.

Eye-tracker data and usability studies

Eye movements are required to scan our visual surroundings. During a visual search the eye movement comprises of fixations, when the eye is fixated on a point, and saccades, the rapid and discrete eye movements between fixations. The area in which there is high visual acuity is rather small, approximately 2 degrees around the centre of the fixation.

Data generated by an eye-tracker also comprises of fixations and saccades and most devices automatically process the data to identify saccades and fixations. For an overview of various algorithms to identify fixations and saccades see Salvucci and Goldberg (2000).

For analysing data generated by an eye-tracker, several metrics are potentially informative. An overview is given here:

- Number of fixations: A large number of fixations generally implies a less efficient search (Jacob and Karn, 2003). Also, a large number of fixations on a particular area of the screen can be indicative of high interest in that area.

- Fixation duration: Long fixation duration often means the user has difficulty extracting information (Jacob and Karn, 2003). A long fixation on an item without any other user action (such as selecting the item) can mean the person is unsure whether the item is actually the target item. For example, in the case of window switching, not all application/window icons are equally informative (see example in Figure 7.12).

- Number of saccades: More saccades indicate a larger amount of visual search.

- Scanpath length: A scanpath is the total length of the saccade-fixation-saccade sequence. Ideally, the scanpath is a straight line to the target, followed by a short fixation on the target (Goldberg and Kotval, 1999). A long scanpath could mean the user is searching for an item.

- Scanpath duration: Scanpath length by itself does not reveals whether the length is caused by long search or a long time spent processing information. Scanpath duration can provide an indication of the complexity of the processing.

- Scanpath convex hull: The scanpath convex hull is the smallest convex polygon that contains the entire scanpath (Goldberg and Kotval, 1999). This gives information about the area covered during the search. The convex hull does not necessarily relate to the length or duration of the scanpath. A lot of searching can take place in a small area, or conversely, a short search across a large area can occur.

- Ratio of fixations on the target to the total number of fixations: A low ratio indicates that the search is not very efficient.

- Saccade direction changes: A direction change larger than 90 degrees could imply that the user's goals have changed or the user interface is not the way the user expected (Poole and Ball, 2005).

Figure 7.12: Two icons that are not equally informative. While the icon on the left could signify *any* kind of file folder, the icon on the right signifies the content of the folder.

When using an eye-tracker for usability studies, several attributes of the eye-tracking device determine its usefulness. Setting the correct time threshold for a fixation can be problematic (Poole and Ball, 2005). Too low a threshold implies many false positives, i.e., identifying too many fixations. It is usually advised to set the lower threshold at at least 100 ms (Poole and Ball, 2005). Furthermore, the sampling rate of the eye-tracker device should not be too low, as too low a frequency means events such as fixations are missed. In general, for usability studies, a sampling rate of 60 Hz is sufficient (Poole and Ball, 2005). Lastly, the degrees of accuracy is of importance, with most eye-tracker being only accurate to within one degree of the actual point of interest (Poole and Ball, 2005). An important fact here is that a person's attention can be directed up to one degree away from the actual point in space where a person is looking without the user moving their eyes (Jacob and Karn, 2003).

Layouts

Three different layouts were used in the experiment: a random layout, where all item locations were randomly assigned with each update of the layout, and Hilbert and Moore treemaps (see Chapter 8), in which items stay in relatively stable locations.

Hardware

The experiment used a *Tobii T60* eye tracker with the 'Tobii Fixation Filter', which defines a fixation as gaze points within a 35 pixel area. This is a good filter for calculating scanpath length, but less suitable for a number of fixations analysis, as it will gives a lot of 'false hits', i.e., identifying something as a fixation while it is not. The alternative is a filter that requires a minimum dwell time before it will log a fixation. This is less suitable for analysing scanpaths as rather continuous 'flowing' scanpaths can be missed as only the fixation points at the start and the end of the scanpath are recorded as a fixation. Pilot testing revealed the 'Tobii Fixation Filter' to be the most suitable for the purpose of the current study.

Procedure

The design and procedure were similar to the experiment in reported in Section 7.4.1. Participants completed trials in a 30-item grid and items were cued following a skewed (Zipfian) distribution of targets; one item was cued 15 times, one item was cued 8 times, followed by 5, 4, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1, and 1. The different layouts were presented in random order.

Design

The scanpath duration dependent measure was analysed using a 3×3 RM-ANOVA for factors *layout* (Hilbert, Moore, random) and *experience* (low, medium, high). Experience was determined by assigning first-time icon selections as low experience, 2^{nd}-10^{th} selections as medium experience, and 12^{th}-15^{th} as high experience. Scanpath duration was used as a measure because (1) the data produced by 'Tobii Fixation Filter' lends itself most to scanpath analysis and (2) while scanpath length could potentially be an informative measure as well, data analysis revealed that, despite best efforts, the eye-tracker regularly 'lost' the eye while it was moving, leading to scanpath lengths that seemed shorter than they actually were. The start and end of the scanpath, where the eye was relatively stationary, was recorded with a high success rate.

Participants

Five students volunteered for the experiment (4 male, 1 female, 21-28 years old). Their participation lasted approximately 15 minutes (including calibrating the eye-tracker for each participant).

Results and discussion

There was a significant main effect for *experience* ($F_{2,8}$=72, p<.001), but not for *layout*, nor was there a *layout*×*experience* interaction (see Figure 7.13).

Similar to the experiment in section 7.4.1, the results show that the random treemap layout supports expertise development, which suggests that guided search takes place. The absence of a statistically significant difference between the layouts and a *layout*×*experience* interaction can most likely be attributed to the small sample size and the variability of the results. Both the variability within subjects and between subjects (as shown in Figure 7.13) is quite large. Large between-subject differences are very common in eye-tracker studies (Goldberg and Wichansky, 2003) and large within-subject differences have also been observed in previous eye-tracker studies (e.g., Byrne et al., 1999). Therefore, there is reason to believe that the absence of a statistically significant difference between the layouts is a consequence of the methodology used. The *trend* of the results, however, is the same as in the empirical study presented in section 7.4.1; the random layout performs worse than the more stable layouts on the medium experience level, but the difference disappears on the high experience level. This is indicative of guided search: on the *medium experience* level the size difference between items is minimal, hence guided search is not facilitated for the random treemap. On the *high experience* level, when the size difference between items is substantial, guided search is facilitated for the random treemap and the differences between the various layouts disappear.

7.5 Qualitative study

Testing of software by end-users is an indispensable tool for researchers (Holzinger, 2005). Therefore, a beta version of SCOTZ was given to five volunteers, who were asked to use it for at least several days. By giving SCOTZ to users while it was

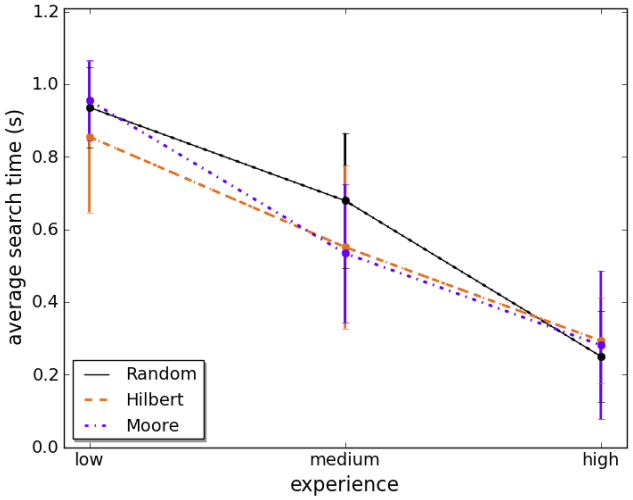


Figure 7.13: Experiment mean search times for the three *layouts* by *experience* including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

still in the beta stages of development, the general concept could be validated and performance problems could be eliminated in an early stage of design (Dolan and Matthews, 1993). After several days of using SCOTZ, participants were given a questionnaire and were interviewed. Questionnaires and interviews are a good way of assessing the subjective satisfaction of users and interviews in particular are of great value because they can be adjusted 'on the fly' to elaborate more about new and unexpected issues that arise during the course of the conversation (Holzinger, 2005).

Based on the results, the following five observations were made:

1. *People did not notice the slight location changes of application zones when using the spiral treemap layout.* Participants commented on never noticing *any* location changes of the application zones when the spiral treemap layout was used, even though gradual changes will have occurred as zones grew and shrunk.

2. *No clear preference for either the spiral or the squarified treemap layout.* The spiral layout was the default layout for SCOTZ, but some participants

did (temporarily) switch to the squarified layout. However, there is no consensus on which layout is preferred. Some participants regarded the squarified layout as being too unstable (i.e., the application zones move too much), while others really liked the squarified layout and clearly preferred it over the spiral layout.

3. *Even a user that mainly used Alt+Tab appreciated the size morphing of the application zones.* Though the size morphing of the application zones does not seem to have direct benefits for people that mainly use Alt+Tab for switching between windows, one participant commented that it helped him to guide attention towards the application he was aiming for.

4. *The application launching functionality was only used by some participants, but it did not bother those who did not use it.* The option to use SCOTZ as an application launcher was only used by some users, yet this functionality did not bother non-users. If anything, retaining the application zones when no windows are open enhances the spatial stability of SCOTZ's layout.

5. *Overriding existing mappings such as Alt+Tab is useful, but risky.* Because SCOTZ was bound to Alt+Tab by default and SCOTZ retained Alt+Tab's (z-order) functionality, SCOTZ could be used without any additional learning. However, cycling to the correct application using SCOTZ (instead of clicking on the zones/thumbnails with the mouse) can be confusing, because it is harder to keep track of the selected item (also see Figure 6.5). Possible solutions are (1) mapping SCOTZ under another key combination, (2) not retaining the z-ordering, but picking an ordering that matches the layout of SCOTZ better, or (3) providing better feedback on the z-ordering (e.g., with a small strip at the bottom of the screen showing the full order).

7.6 Lab study

Sections 7.2, 7.3 and 7.4 provided theoretical and empirical validation of the design principles of SCOTZ, but it is yet unclear how SCOTZ compares to other (common) window switching interfaces. Therefore, this section presents and empirical evaluation of SCOTZ and the window switching interfaces available in

the most recent version of the widely used Microsoft Windows operating system (Windows 7): the Taskbar and Alt+Tab.

The Windows 7 Taskbar and Alt+Tab were chosen for comparison because (1) Microsoft Windows is the most commonly used operating system, and therefore a comparison with the tools available in Windows 7 is relevant, and (2) these two tools present a *challenging* condition to compare SCOTZ against in terms of the key design principles of SCOTZ. The Windows 7 Taskbar places application icons in stable locations (unless a program is closed and re-opened) and Windows Alt+Tab provides explicit support for switching back to recently used windows because it places (the first six) window representations by their place in the z-ordering of windows, which is very similar to a recency ordering.

7.6.1 Method

In the experiment, participants were presented with a set of windows in Microsoft Windows 7, such as a word processor with several documents open, an e-mail application, a game, a video player, etc. Participants completed a series of tasks in which they had to switch to a particular window using the Windows 7 Taskbar, Windows 7 Alt+Tab, or SCOTZ in a successive series of tasks (see Figure 7.14). In each condition, participants were instructed to use one particular window switching tool *exclusively*.

Some windows were prompted often, while other were hardly ever prompted, following the findings reported in Chapter 5: 80% of window switches was to 35% of all windows.

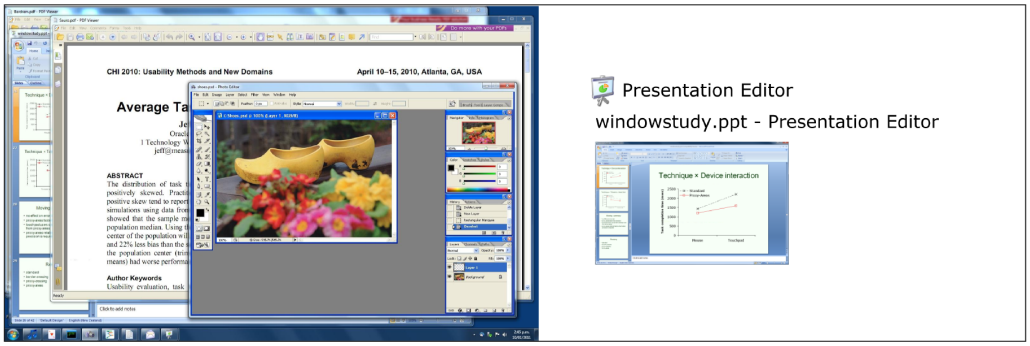


Figure 7.14: The experimental interface: all windows are on the primary screen on the left, and the current task is on the secondary screen on the right.

7.6.2 Procedure

At the start of each condition participants were given a verbal explanation and a demo of the window switching tool used in that particular condition. Participants then performed a series of 20 practice tasks with the window switcher before starting the experimental tasks.

At the start of each task, all windows were temporarily hidden from view and the user was prompted to press a 'Next' button at the centre of the secondary screen. Pressing this button revealed the next target window on the secondary screen (by showing the application icon, the window title and a window preview thumbnail of the target). Next, participants were prompted to click a 'Start' button in the centre of the primary screen, after which all the windows were unhidden, and participants then had to switch to the target window. If the participant switched to the incorrect window nothing happened. In total, participants performed 80 tasks in each condition (excluding the practice tasks).

After each condition, the participant filled out a short questionnaire regarding the window switching tool that had just been used.

7.6.3 Design

Switching time and *errors* (switching to a non-target window) were measured across three levels of the independent variable *interface* (Taskbar, Alt+Tab and SCOTZ) and analysed using a one-way repeated-measures ANOVA. The experiment used a within-subject design and the order in which the conditions were presented to the participants was counterbalanced.

7.6.4 Software and hardware

All the content of the windows used in the experiment was non-modifiable to minimise distraction, to prevent accidental interaction with the windows, and to allow for consistent window previews in the window switching tools throughout the tasks. To prevent learning effects across conditions, three different window sets were generated, all with unique applications and windows. The window sets were counterbalanced across the three conditions. Each window set contained 8 applications and 15 windows. For example, one of the window sets contained a PDF reader (with 4 windows open), a photo editor (3 windows), a presentation editor

(2 windows), an HTML editor (2 windows), a music player, an email application, a command prompt, and a card game.

Window icons that are already in use by well-known applications were not used, to ensure that all participants started off with equal knowledge of the application icons. This is particularly important for the evaluation of the Windows 7 Taskbar, which shows only application icons.

The full-screen version of SCOTZ with the squarified treemap algorithm was used. All application zones in SCOTZ were fixed (i.e., did not change during the experiment) and were pre-set to reflect the frequencies with which applications/windows were switched to during the experiment.

A mouse with an extra side button was used in the experiment, and this side button was used to invoke SCOTZ.

7.6.5 Questionnaires

The experiment used the NASA Task Load Index[6] (NASA-TLX) to assess the perceived workload in each of the conditions. The NASA-TLX questionnaire contains six sub-scales, each related to a particular aspect of perceived workload: mental demand, physical demand, temporal demand, performance, effort and frustration. Participants are requested to rate their agreement on six 7-point Likert scales:

- Mental demand: How mentally demanding was the task? (Very Low . . . Very High)

- Physical demand: How physically demanding was the task? (Very Low . . . Very High)

- Temporal demand: How hurried or rushed was the pace of the task? (Very Low . . . Very High)

- Performance: How successful were you in accomplishing what you were asked to do? (Perfect . . . Failure)

[6] http://humansystems.arc.nasa.gov/groups/TLX/downloads/TLX.pdf

- Effort: How hard did you have to work to accomplish your level of performance? (Very Low ... Very High)

- Frustration: How insecure, discouraged, irritated, stressed, and annoyed were you? (Very Low ... Very High)

Two more questions were added to assess the perceived ease of learning to operate the window switcher (*operation*) and the perceived ease of learning window locations (*location learning*) in the window switcher:

- Operation: How easy was it to learn to operate this window switcher? (Very Easy ... Very Difficult)

- Location learning: How easy was it to learn the locations of the windows in this window switcher? (Very Easy ... Very Difficult)

Finally, participants were asked to rank the three window switching interfaces from most to least preferred.

7.6.6 Participants

Twelve people, all university students, participated in the experiment. Age ranged from 20 to 35 years old (mean 27); two participants were female. All participants were experienced computer users: computer use was at least 30 hours per week for each of the participants. Participants were reimbursed with a NZ$10 shopping voucher. The experiment took approximately 40 minutes to complete.

7.6.7 Results and discussion

Selection times

The mean time to switch to a window for each of the methods is shown in Figure 7.15. The results for the Windows 7 Taskbar are split by Taskbar button (for applications with only one associated window) and Taskbar thumbnail (for applications with more than one associated window, where the user first has to select the application icon and *then* the window in the fanned out sub-menu).

Mean window switching times when using a Taskbar button, a Taskbar thumbnail, Alt+Tab and SCOTZ are 1.1s, 2.1s, 2.1s and 1.2s, respectively, giving a significant effect of *interface*: $F_{3,33}$=53, $p<.001$. Post hoc analysis (Bonferroni correction, α=.05) reveals pairwise differences between all tools (all $p<.001$) except the Taskbar button and SCOTZ, and the Taskbar thumbnail and Alt+Tab.

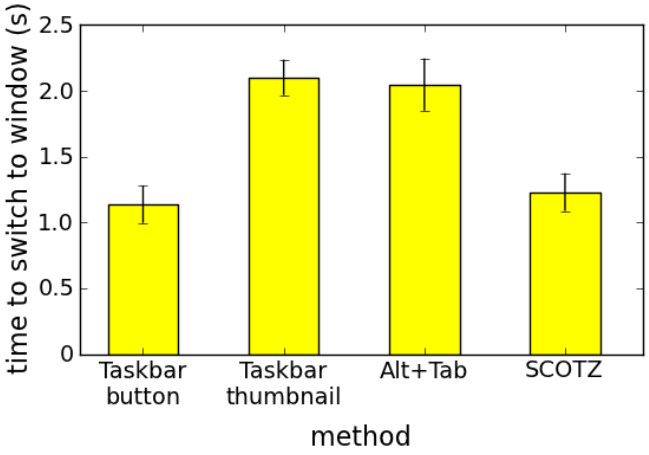


Figure 7.15: Window selection times for the various window switching tools including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

A more detailed analysis of window switching times when Alt+Tab is used is shown in Figure 7.16, which shows the selection times for Alt+Tab ranked by position of the target window in the Alt+Tab ordering split by input method (using the keyboard to sequentially step through the list of thumbnails, or using a mouse to click on the target thumbnail). By design, some of the target windows in the experiment were high up in the Alt+Tab ordering and others further down, following a nearly uniform distribution across all possible positions.

Three observations are apparent from Figure 7.16: (1) window selection time when using Alt+Tab with mouse input is relatively constant across positions of the target thumbnail, (2) window selection time when using Alt+Tab with keyboard input increases linearly as the position of the target thumbnail in the list of windows becomes higher (r=.963, $p<.01$), and (3) Alt+Tab with keyboard input is very efficient for switching back to the previously used window (position 1 in the ordering); the mean window switching time is 0.9 seconds for this particular

type of window switch, which is shorter than the mean switching times for both the Taskbar and SCOTZ. However, this performance benefit quickly disappears when the target window is further down the list of windows.

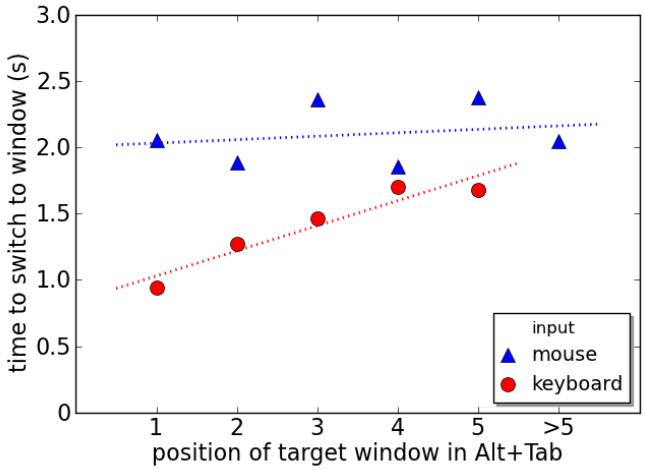


Figure 7.16: Window selection times for Alt+Tab sorted by position of the target window in Alt+Tab and split by mouse and keyboard input. Participants almost never used the keyboard to select a window further than position 5 in the Alt+Tab ordering, hence there is no (reliable) data for this value.

Error rates

Figure 7.17 shows the error rates for the window switching tools used in the experiment. Mean error rates for Taskbar button, Taskbar thumbnail, Alt+Tab and SCOTZ are 0.8%, 5.8%, 2.8% and 2.7%, respectively. The difference between these error rates is significant (RM-ANOVA, $F_{3,33}$=5, p<.01). Post hoc analysis (Bonferroni correction, α=.05) reveals a pairwise difference between the Taskbar button and Taskbar thumbnail (p<.05).

Subjective measures

The NASA-TLX worksheet (see Figure 7.18) results showed significantly different ratings for *mental demand*, *effort*, *location learning*, *frustration* (Friedman test, all p's <.001), and *operation* (p<.01). Post hoc pairwise comparisons (Bonferroni correction, α=.05) reveals significant differences between Alt+Tab and

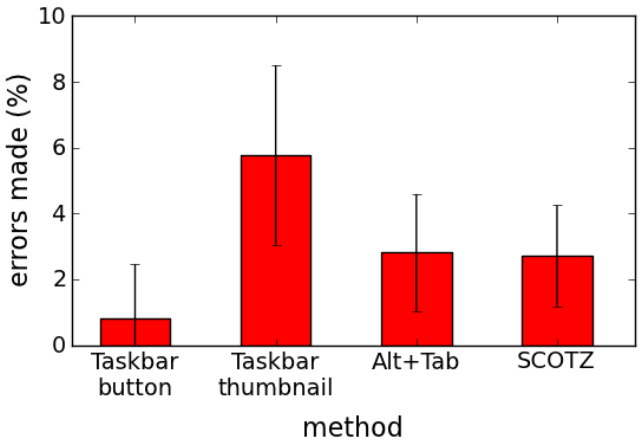


Figure 7.17: Percentage of errors made with the various window switching tools including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

SCOTZ on all five aforementioned factors, between the Taskbar and Alt+Tab on all these factors except *frustration*, and between the Taskbar and SCOTZ on the *mental demand* and *location learning* factors.

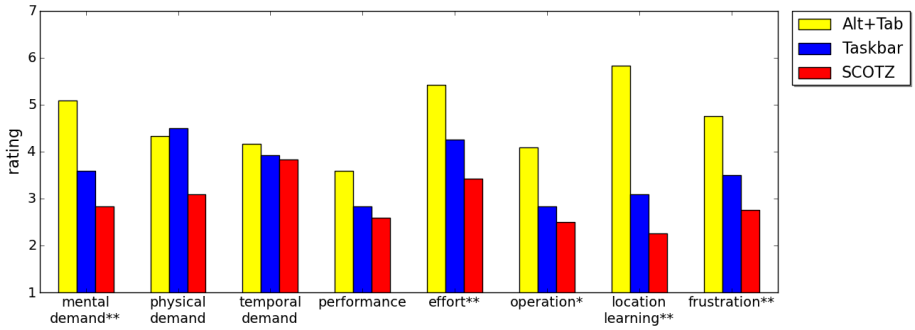


Figure 7.18: Questionnaire results; lower ratings are better. * Difference is significant, $p<.01$. ** Difference is significant, $p<.001$.

All participants preferred SCOTZ the most, and 9 out of 12 participants preferred Alt+Tab the least and 3 out of 12 participants preferred the Taskbar the least.

Discussion

Window switching times. The findings provide evidence that SCOTZ is faster than both Taskbar thumbnails and Alt+Tab. The study finds no significant difference between the average window switching time using SCOTZ and a Taskbar button. However, Taskbar buttons are not available when there is more than one window associated with the application (forcing users to resort to Taskbar thumbnails). The study presented in Chapter 5 showed that 53.3% of windows belong to an application that has more than one window associated with. In other words, the majority of windows can only be reached via a Taskbar thumbnail, which is significantly slower than SCOTZ.

Subjective measures. All participants ranked SCOTZ as the most preferred tool. Users perceived SCOTZ as less mentally demanding, costing less effort, and less frustrating than Alt+Tab, as well as finding it easier to learn to operate and to learn item locations in SCOTZ compared to Alt+Tab. Also, window locations in SCOTZ were perceived as easier to learn than those on Taskbar and SCOTZ was perceived as less mentally demanding than the Taskbar. These two factors are likely related: SCOTZ is less mentally demanding because it is easier to learn locations, thereby reducing the cognitive burden for users. It is interesting that users found locations in SCOTZ easier to learn than locations of items on the Taskbar, as in both cases these were completely stable in the current study.

Alt+Tab. Overall, Alt+Tab was unpopular, with 75% of participants ranking it as least preferred. Alt+Tab was also judged to be more mentally demanding and costing more effort than the Windows Taskbar. Last, users found it harder to learn to operate and learn item locations in Alt+Tab than with the Taskbar. One participant commented that he/she "*hated how Alt+Tab icons moved around*". However, these results for Alt+Tab might have been negatively influenced by the fact that users had to use Alt+Tab for *all* window switches in the experiment. The results show that Alt+Tab is very efficient for switching back to the most recently used window, with this particular type of switch outperforming both the Taskbar and SCOTZ. One participant commented "*I use Alt+Tab to switch between the most recent windows, and other methods for older windows.*" Such a 'mixed approach' (using Alt+Tab to switch back to the most recently used window, but another method for other types of window switches) might lead to higher user

satisfaction than the 'enforced' use of Alt+Tab for all window switches that was the case in the experiment.

Comparison to other window switching tools. The experiment compared user performance with SCOTZ against that with the Windows 7 Taskbar and Alt+Tab. As reviewed in Chapter 2, many other window switching tools exist. However, the comparative success of SCOTZ can probably be attributed to its spatial stability and predictability of item locations. Unfortunately, hardly any of the tools discussed in Chapter 2 section exhibit such stability and/or predictability. As the positioning of window representations in Exposé, for example, is related to the positions of the actual windows Exposé's representation of available windows is altered whenever windows are moved, opened, or closed, so users can never be certain where their target windows will appear. Task-based approaches, which are common, present similar problems. In general, task-based approaches are, by design, not compatible with placing application and window controls in stable positions, as tasks are of a transient nature. Also, the predictability of locations of the window representations is heavily dependent on the method used for identifying tasks, with automatic grouping being risky as this can be unpredictable.

7.7 Conclusion

In this chapter, two empirical studies first showed that (1) spatially stable layouts allow faster acquisition than recency and random layouts for skewed distributions such as those occurring in window switching tasks, and (2) that the instability inevitably caused by size morphing does not severely impact user performance compared to the (idealistic but impractical) gold standard of absolute spatial stability. The results of the study presented in section 7.4.1 reveal that the slight instability of the spiral and squarified layouts does impair performance, but the effect seems to be weak and therefore hard to detect.

Next, theoretical examination of the effect of size morphing demonstrated that size morphing leads to an overall performance advantage because of the Fitts' Law targetting time advantage of increasing item size. The results in sections 7.3.1 and 7.3.2 show that size morphing can lead to an improvement of Fitts' Law targetting time. In other words, the improved Fitts' Law targetting times of larger items outweighs the worsened Fitts' Law targetting times of the smaller items,

because the larger items are selected more often (hence they are larger than their less 'popular' counterparts). The more pronounced (or skewed) the data distribution, the more beneficial size morphing can be. Not all treemap layouts lead to an improvement of Fitts' Law targetting time, with the aspect ratio of the items in the treemap being a directly contributing factor to this difference (the Fitts' Law targetting time benefit is strongest for layouts with perfectly square items).

An empirical and an eye-tracker study demonstrated that size morphing facilitates finding items. The results of the studies in sections 7.4.1 and 7.4.2 are quite similar: the random layout performs worse than the other layouts on the medium experience level, but this difference disappears on the high experience level (see Figures 7.11 and 7.13). In other words, even the random treemap supports expertise development and this can be explained by guided search.

The results of these studies support the foundations of SCOTZ's design and help to better understand the comparative success of SCOTZ.

The results from the studies using a random treemap are by no means intended to advocate letting go of the stability principle. Rather, the positive effects of size morphing on finding items can serve as 'safety net' if the user forgets item locations, e.g., when the user has not used the computer for a longer period of time.

The lab study demonstrated the performance benefits of SCOTZ over two common window switching tools: the Microsoft Windows 7 Taskbar and Alt+Tab. This study also generated valuable insights regarding the most recent window switching tools available in Microsoft Windows 7.

More research into the suitability of SCOTZ for Alt+Tab users could shed more light on how these users can best be supported in their window switching activities. Interestingly, even Alt+Tab users reported to benefit from the size morphing of the application zones in SCOTZ, but retaining the Alt+Tab order in SCOTZ did confuse these users. Ideally, SCOTZ should retain the rapid back-and-forth switching between two windows that Alt+Tab offers while also assisting users in finding items further down the Alt+Tab ordering.

Chapter VIII

Enhanced Spatial Stability with Hilbert and Moore Treemaps

Chapter 6 introduced the new window switcher SCOTZ, which employs treemaps for the layout of the application zones. The results in Chapter 7 revealed that spatial stability is important, but, interestingly, no existing treemap algorithms are stable when data updates, *and* when items are added/deleted, *and* when many changes have taken place (i.e., the cumulative effect of data changes). In other words, some treemaps are relatively stable when items are added, but not when data is updated, other treemaps are relatively stable from individual update to update, but these changes add up to large cumulative changes after many updates, and so forth. This chapter focuses on the formal evaluation of the suitability of various treemaps for SCOTZ, particularly in terms of spatial stability.

This chapter first demonstrates that stability is not fully captured by the commonly used 'distance change' metric (e.g., Bederson et al., 2002; Shneiderman and Wattenberg, 2001; Tu and Shen, 2007). Some treemaps with relatively low distance change values from update to update can nevertheless be relatively unstable as the underlying data changes because items never 'settle' in a particular area of the screen. To resolve this problem the 'location drift' metric is introduced, that is designed to better encapsulate stability over time.

Next, this chapter introduces two treemaps that perform well across all metrics of spatial stability, including the new 'location drift' metric, while also performing well in terms of other common treemap metrics, such as aspect ratio. These new treemaps are based on Hilbert and Moore space filling curves (Sagan, 1994).

Finally, as this chapter introduces a novel measure of spatial stability (location drift), an empirical study manipulates the spatial properties of various space-filling layouts, in order to examine how the distance change and location drift metrics influence actual user performance in item retrieval.

8.1 Contributions and findings

- an overview of which metrics can be used to assess the stability of treemap layouts;
- a mathematical formalisation of existing treemap metrics;
- introduction of the 'location drift' metric;
- introduction of the Hilbert and Moore treemaps;
- a theoretical comparison of the Hilbert and Moore treemaps and various other treemaps;
- an empirical study demonstrating that 'location drift' is a useful metric for evaluating treemap stability.

8.2 Treemap algorithms

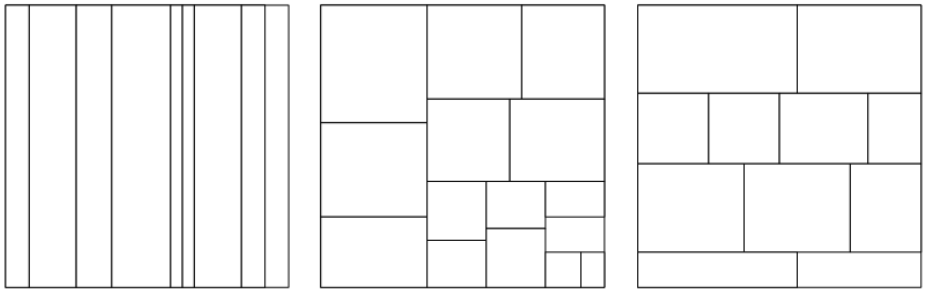
A treemap is a space-filling structure for the visualisation of data (Johnson and Shneiderman, 1991). It recursively divides an area into rectangles of various sizes, with the size of the rectangles representing some underlying quantitative data attribute. The benefits of treemaps are manifold. First, by having the size of items in the treemap reflect the underlying data large data sets can be communicated in an effective manner (Bruls et al., 2000). Second, as treemaps are space-filling no (precious) screen estate is wasted (Johnson and Shneiderman, 1991). Third, hierarchies are clearly identifiable because of the nested structure of a treemap. Hierarchies can be emphasised more by the use of colour (for example, see Figure 8.1).

The applications of treemaps are diverse. For example, treemaps have been used to visualise information about the stock market (Wattenberg, 1999), as a photo browser (PhotoMesa) (Bederson et al., 2002), to visualise threaded discussion forums on PDAs (Engdahl et al., 2005), depicting hierarchical structures (Shi et al., 2005; Stasko, 2000), visual decision making (Asahi et al., 1995) and the analysis of social networks (Frantz and Carley, 2005).

Figure 8.1: Example of a treemap generated by the website www.newsmap.jp. Colour signifies the news category, size reflects the number of related articles.

Many alternative treemap algorithms have been proposed since their introduction (Johnson and Shneiderman, 1991), offering different advantages in different contexts. The early 'slice and dice' algorithm (Shneiderman, 1992) (see Figure 8.2a) recursively slices the available space into parallel rectangles, but in doing so it is susceptible to producing items with high aspect ratios, which is undesirable as it makes items hard to label and select. Squarified (Bruls et al., 2000) (see Figure 8.2b) and cluster algorithms (Wattenberg, 1999) aim to produce lower aspect ratios. However, the squarified treemap highlights another property of treemaps that is important: distance change. The distance change metric quantifies how much the positions of the items in the treemap change when the underlying data is changed. Because the squarified algorithm sort items by size, small changes in the data can mean items move quite a large distance.

One way to ensure treemap layouts are relatively stable when the underlying data is changed is to preserve the underlying ordering of the data. Ordered treemap layouts preserve the underlying data order, with example algorithms including slice and dice, pivot (Shneiderman and Wattenberg, 2001), strip (Bederson et al.,



(a) Slice and Dice (b) Squarified (c) Strip

Figure 8.2: Examples of treemaps generated by several algorithms.

2002) (see Figure 8.2c), and spiral layouts (Tu and Shen, 2007). However, some of these algorithms do come with their own individual disadvantages. For example, when the strip treemap is used rather sudden changes in the layout can occur when an item moves from the right side of the treemap to the left side (on the next strip). This problem is addressed by the spiral treemap (Tu and Shen, 2007), where items are arranged following a spiral structure.

Many other designs have been proposed to meet the requirements of specific application areas. For example, Cushion treemaps (van Wijk and van de Wetering, 1999) are designed to highlight hierarchical structure and Quantum treemaps (Bederson et al., 2002) are designed to accommodate items with a fixed size, such as photos.

By definition, treemaps have rectangular items: "*[a treemap is a] two-dimensional (2-d) space-filling approach in which each node is a rectangle whose area is proportional to some attribute such as node size*" (Shneiderman, 1992, p. 92). However, this can lead to situations where the aspect ratios of the items are unavoidably unbalanced, as in the 'extreme' example when there are two items with respective weights of 999 and 1 (Wattenberg, 2005). To resolve this issue, some newly developed algorithms create a layout with non-rectangular items. For example, Voronoi treemaps (Balzer et al., 2005) use arbitrary polygons rather than rectangles for visualising software metrics, and the Jigsaw layout (Wattenberg, 2005) uses the space-filling H-curve (Niedermeier et al., 1997) to create non-rectangular puzzle-piece shaped items (the new treemaps presented in this chapter use the space-filling Hilbert and Moore curves). Wattenberg (2005) also provides a math-

ematical analysis of the use of space-filling curves for supporting space-filling visualisations (such as treemaps). The use of non-rectangular items, however, does not come without its disadvantages. Wattenberg (2005, p. 27) observes that "*[...] irregular puzzle-piece shapes certainly look odd, and seem likely to make it more difficult to compare areas and understand the tree topology than a treemap [with rectangular items] does*".

8.3 Treemap stability

The stability of the treemap that is used for the layout of the application zones in SCOTZ is important as stable layouts lead to good performance, while unstable layouts do not (see results in Chapter 7). The stability of a treemap is often expressed as the **distance change** of items when the treemap data is updated (e.g., Bederson et al., 2002; Tu and Shen, 2007), with lower distance change values indicating a more stable layout. If an item *i* in a treemap is defined as the rectangle (*x*, *y*, *w*, *h*), with *x*, *y* the position of one of the corners and *w* and *h* the width and the height, the distance change between two positions can be calculated and averaged for all items using Equation 8.1 (Bederson et al., 2002).

$$\overline{distance\ change} = \frac{1}{n}\sum_{i=1}^{n}\sqrt{(\Delta x_i)^2 + (\Delta y_i)^2 + (\Delta w_i)^2 + (\Delta h_i)^2} \qquad (8.1)$$

Distance change variance (Tu and Shen, 2007) is sometimes used to complement the distance change metric; a low mean distance change, but a high distance change variance means a few items move a lot. Distance change variance is calculated using Equation 8.2.

$$dist.\ change\ variance = \sum_{i=1}^{n}(distance\ change_i - \overline{distance\ change})^2 \qquad (8.2)$$

Although the distance change metric provides important insights into the stability of a treemap, it does do not describe the instability caused by 'drifting' item positions. For example, consider the two situations shown in Figure 8.3: in the left layout an item flips between position A and B, while in the right layout the

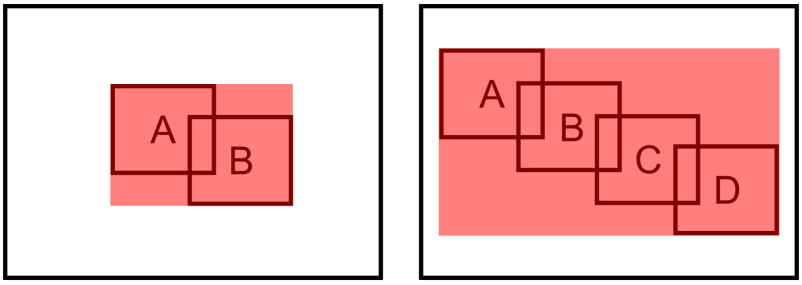


Figure 8.3: Example of items that move the same distance, but have different location drift. Both images show different positions of an item and the associated footprint. Left: low location drift and a small footprint because the item flips between two positions A and B. Right: high location drift and large footprint because the item moves from position A to B, from B to C, and from C to D.

item drifts from A to B to C to D, and so forth. If these positions are equidistant distance change is identical for all updates, but the high cumulative movement distance of the right layout is likely to have a stronger detrimental effect on retrieval because the item never 'settles' in one area of the screen.

This chapter introduces the **location drift** metric to encapsulate the instability caused by drifting item positions. The underlying assumption motivating the location drift metric is that human performance in item retrieval is facilitated when items are located in a particular spatial region of the display, and hindered when locations gradually drift. Location drift is calculated by analysing the item 'footprint', which is the rectangular area defined by the extreme border locations of an item across all location changes (see Figure 8.3). If the top left of the screen is defined as coordinates (0,0), x_{min} is the x-coordinate of the left border of the left-most position the item has ever been at, y_{min} the y-coordinate of the top border of the top-most position the item has ever been at, x_{max} the x-coordinate of the right border of the right-most position the item has ever been at and y_{max} the y-coordinate of the bottom border of the bottom-most position the item has ever been at. The footprint is the area described by these coordinates, and location drift is the ratio of the footprint to the mean item size, see Equation 8.3.

$$\overline{location\ drift} = \frac{1}{n}\sum_{i=1}^{n}\frac{(y_{max}-y_{min})_i \times (x_{max}-x_{min})_i}{\overline{size_i}} \tag{8.3}$$

8.4 Other treemap metrics

Three other metrics (not related to layout stability) are commonly used for evaluating and comparing treemap algorithms: aspect ratio, readability and continuity.

The **aspect ratio** of an item is the ratio of the longer dimension to the shorter one. Treemaps with a high mean aspect ratio are undesirable, because items are hard to recognise, select, and label, as well as being visually unattractive. The mean aspect ratio of a treemap is calculated using Equation 8.4 and values range from 1 (treemaps with perfectly square items) to very high (treemaps with long and 'stretched' items).

$$\overline{aspect\ ratio} = \frac{1}{n}\sum_{i=1}^{n} max(\frac{width_i}{height_i}, \frac{height_i}{width_i}) \tag{8.4}$$

The **readability** metric (Bederson et al., 2002) measures how easy it is to visually scan a treemap layout, and quantifies this by calculating the number of times the reader's gaze must change direction when scanning a treemap in order. This readability metric is in the range [0,1], with 0 signifying very poor readability and 1 maximal readability. Readability is defined mathematically in Equation 8.5, from Bederson et al.'s statement:

> *"[...] we consider the sequence of vectors needed to move along the centers of the layout rectangles in order, and count the number of angle changes between successive vectors that are greater than .1 radians (about 6 degrees). To normalise the measure, we divide this count by the total number of rectangles and then subtract from 1."* (Bederson et al., 2002, p. 842)

$$readability = 1 - \frac{|angle\ >\ .1\ radians|}{n} \tag{8.5}$$

Similar to readability, **continuity** (Tu and Shen, 2007) quantifies how easy it is to visually scan a treemap layout. Continuity is calculated by counting how many items that are consecutive in the data ordering are adjacent in the treemap, see Equation 8.6.

$$continuity = \frac{|adjacent\ consecutive\ items|}{n-1} \tag{8.6}$$

8.5 Hilbert and Moore treemaps

The (informal) observation that none of the currently available treemap algorithms produce a very stable layout inspired the development of two new treemap algorithms: Hilbert and Moore. Hilbert and Moore treemaps are intended to perform well across all metrics of stability, including location drift, as well as on the other metrics. Inspired by Wattenberg's analysis of space-filling curves for visualisations (Wattenberg, 2005) these two new treemap algorithms are based on Hilbert and Moore space-filling curves. However, unlike Wattenberg's work, these new treemaps maintain the traditional rectangular item shape. This section first describes the Hilbert and Moore space-filling curves and then the Hilbert and Moore treemap algorithms.

8.5.1 Hilbert and Moore space-filling curves

The Hilbert and Moore treemap algorithms are based on two (similar) space-filling curves (Sagan, 1994; Peano, 1890), shown in Figure 8.5. A space-filling curve is a self-similar continuous curve which completely covers a N-dimensional space without self-intersection. Examples of so-called level 0 and level 1 Hilbert curves (Hilbert, 1891) are shown in Figure 8.4. The level 0 curve evolves to a level 1 curve by applying a standard set of rules, as shown in Figure 8.4. The level 1 curve comprises four quadrants with level 0 curves, albeit some rotated 90 degrees and flipped along the horizontal or vertical axis. In each of these quadrants the level 0 to 1 transition rules can be applied again to create a level 2 Hilbert curve, and so forth.

The Moore curve (Moore, 1900) is a variant of the Hilbert curve with a different level 0 to level 1 transition. This results in a curve where the start and end are adjacent points (rather than adjacent corners, as is the case for the Hilbert curve). The level 0 to level 1 transition (see Figure 8.4) creates an H-shaped curve. Subsequent transitions are the same as the Hilbert curve transitions. Figure 8.5 shows level 2 Hilbert and Moore space-filling curves.

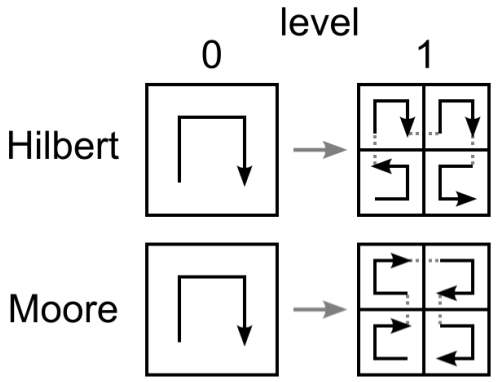


Figure 8.4: Level 0 and 1 Hilbert and Moore curves.

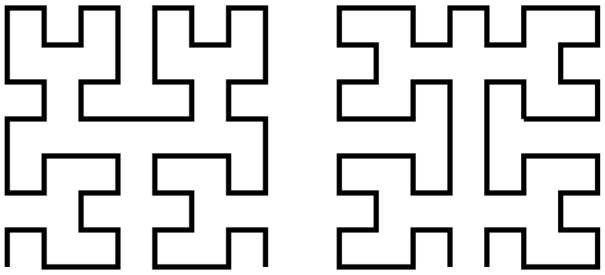


Figure 8.5: Level 2 Hilbert and Moore space-filling curves.

8.5.2 Hilbert and Moore treemap algorithms

To create a treemap based on the Hilbert and Moore space-filling curves the (ordered) data set is recursively divided in four weighted sections, until each section contains four or fewer items. Next, the sections are laid out on the screen. In the last step the actual items are laid out, during which some aspect ratio optimizing is applied. This section demonstrates the algorithm by using the example shown in Table 8.1.

Table 8.1: Ten numbered items and their associated values and sections

item #	1	2	3	4	5	6	7	8	9	10
value	5	5	2	8	3	2	2	3	6	10
section	A	A	A	B	B	C	C	C	C	D

Step 1: Recursively divide list in weighted sections.
First, the algorithm divides the data in four weighted sections, such that each section has a relative cumulative weight of its values that is (approximately) 25%, while the underlying ordering of the data is maintained. This is recursively repeated until each section contains four or fewer items. For the example shown in Table 8.1, this process generates 4 sections. These sections are labelled A, B, C and D in Table 8.1, and have cumulative weights 12 (26%), 11 (24%), 13 (28%) and 10 (22%), respectively.

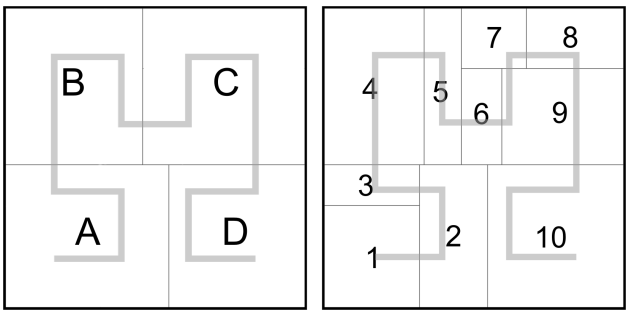


Figure 8.6: Generation of a Hilbert treemap. Left: Layout of the sections. Right: Layout of the actual items. Overlay is a level 1 Hilbert curve.

Step 2: Lay out the sections.
After the list is split, the sections are sequentially laid out on the screen following the directional rules dictated by the Hilbert/Moore curve, see Figure 8.6, left.

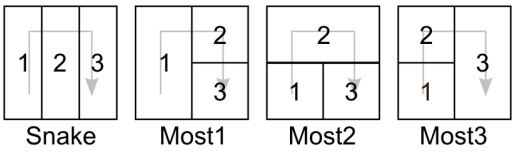


Figure 8.7: Four layout options when the section has three items: Snake, Most1, Most2 and Most3.

Step 3: Lay out the items.
When the actual items are laid out (not the sections), some aspect ratio optimising is applied by comparing several candidate layouts if a section contains three or four items. These candidate layouts all maintain the underlying data order and place the first and the last item in adjacent corners. When a section contains

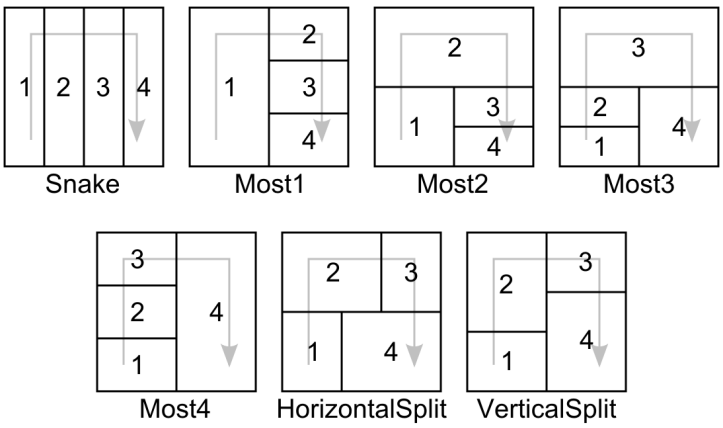


Figure 8.8: Seven layout options when the section has four items: Snake, Most1, Most2, Most3, Most4, HorizontalSplit and VerticalSplit.

three items, the algorithm compares the mean aspect ratio of four layouts called 'Snake', 'Most1', 'Most2' and 'Most3' (see Figure 8.7). When a section contains four items, three additional layouts are evaluated; 'Most4', 'HorizontalSplit' and 'VerticalSplit' (see Figure 8.8). The layout with the lowest mean aspect ratio is used. The layout is placed such that the continuity of the treemap as a whole is maintained; the first item neighbours the last item in the previous section and the last item neighbours the first item in the next section. The resulting treemap is shown in Figure 8.6, right. An example of a slightly larger treemap generated by the Moore treemap algorithm is shown in Figure 8.9.

8.6 Theoretical comparison of treemaps using metrics

This section presents an evaluation of Hilbert and Moore treemaps and several other treemap algorithms using the stability metrics described in Section 8.3 (including the new 'location drift' metric) and the remaining metrics described in Section 8.4. The treemap algorithms that are compared are Hilbert and Moore, slice and dice (Shneiderman, 1992), squarified (Bruls et al., 2000), strip (Bederson et al., 2002), spiral (Tu and Shen, 2007), and three variants of the pivot algorithm (pivot by size, pivot by middle and pivot by split size) (Shneiderman and Wattenberg, 2001).

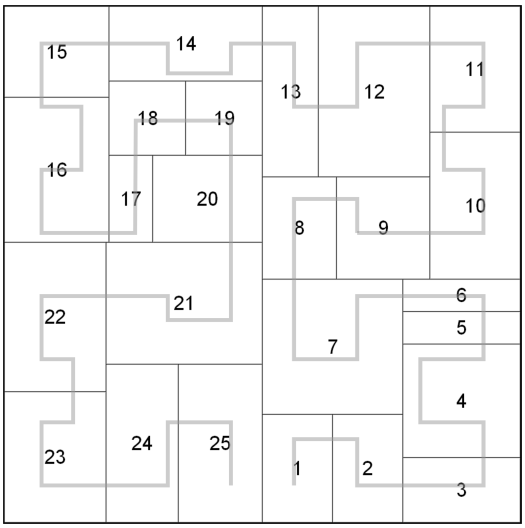


Figure 8.9: A Moore treemap. The grey line denotes the space-filling curve.

To evaluate the metrics of these treemaps a large set of treemaps for a representative data set is required. Multiple trials (i.e., creating a large set of treemaps for each algorithm) are required because ordered treemaps (such as pivot, strip and spiral layouts) will be (visually) different for different orderings of a data set. For example, if the largest item is in the middle of the data set the layout will look different (and have different metric values) than if the largest item is near the front or end of the data ordering. Therefore, to calculate accurate and representative values for the various metrics they should be calculated for multiple data sets (with different orderings).

8.6.1 Method

Three separate 'batches' of simulations were run for (1) the calculation of the stability metrics when data is updated, (2) the calculation of the stability metrics when an item is added to the layout, and (3) the calculation of the remaining metrics (aspect ratio, readability and continuity). The stability metrics for data updates were calculated using a series of Monte Carlo trials to simulate temporally updating data (which is a common procedure, e.g., Tu and Shen, 2007). The effect of item addition was examined because the effect of *adding or deleting items* on treemap stability is not necessarily the same as the effect of *data updates*; some treemap layouts are relatively stable when data updates, but not when items are

added/deleted and vice versa (the effect of item addition on layout stability is the same as the effect of item deletion). Finally, static layouts were used to calculate the other metrics (aspect ratio, continuity and readability).

The data items were assigned values according to a Zipfian distribution (α=1), then randomly ordered: for example, a 10 item dataset might be $\{$*3.33, 1.25, 1.11, 1.43, 10, 2.5, 1.67, 1, 5, 2*$\}$. Zipfian distributions are a common natural data distribution, and often used for treemap evaluations (Bederson et al., 2002). To cover a variety of cases the metrics were calculated for three different set sizes: 10, 30, and 50 items. All calculations apply to a square (1:1) treemap.

Dynamic updates

To measure the effect of data updates on the stability metrics a series of 100 trials (for each set size) of 100 steps each were run, where each step simulates a data update. At the start of each trial, item sizes followed the Zipfian data distribution (α=1). Next, data updates were simulated by multiplying each item with e^x (with x drawn from a normal distribution with mean 0 and standard deviation 0.05) in each step and the average distance change of all items and distance change variance were calculated. Location drift was calculated at the end of each trial. Finally, the metrics were averaged across all trials.

Item addition

For the calculation of the effect of item addition 100 trials (for each set size) were run where, once again, the layout started off with item sizes following a Zipfian data distribution (α=1). Next, a single random item was inserted into the treemap (random location in the dataset and random size between the minimum and maximum of the dataset) and the layout recalculated. After this update, the average distance change of all items and distance change variance were calculated. The metrics were averaged across all trials.

Remaining metrics

For the calculation of aspect ratio, continuity and readability 100 static layouts (for each set size) with item sizes following a Zipfian data distribution (α=1) were generated, and the metrics averaged across all layouts.

8.6.2 *Results*

This section is split in three parts: the results of the stability metrics for data updates, the stability metrics for item addition and the remaining metrics.

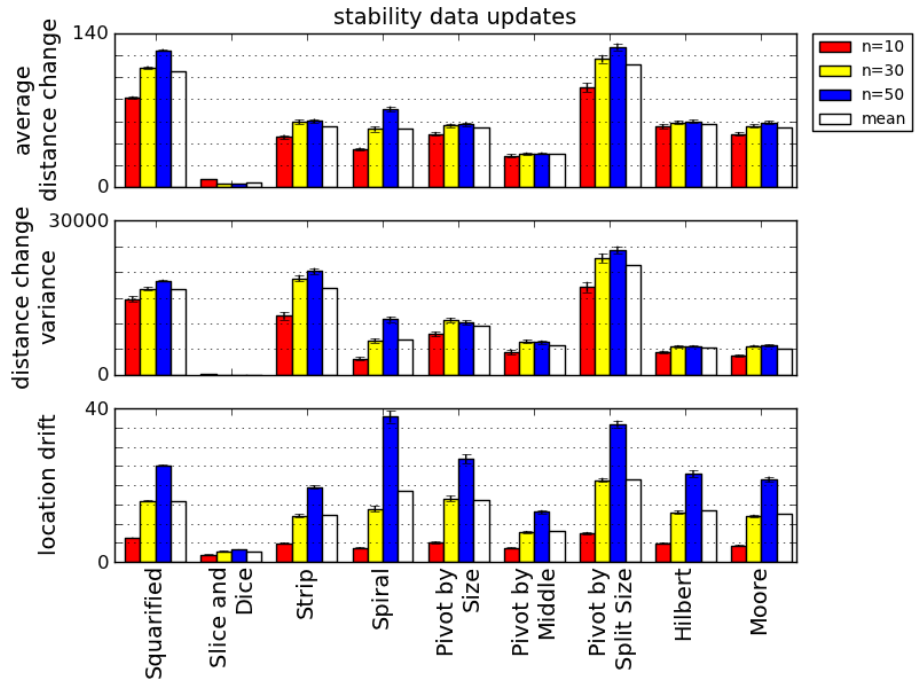


Figure 8.10: Distance change, distance change variance, and location drift metrics for data updates. Error bars represent +/- 1 SE.

Stability - Dynamic updates

Figure 8.10 shows the results of the comparison of different treemap layouts on the distance change, distance change variance and location drift metrics when data is updated.

The first row of Figure 8.10 shows that the average distance change of the Hilbert and Moore treemaps is lower than squarified and pivot by split size, but higher than slice and dice and pivot by middle treemaps.

The second row of Figure 8.10 shows that distance change variance of the Hilbert and Moore treemaps is lower than all other treemaps except slice and dice. Also, the distance change variance for the Hilbert and Moore treemaps is similar

to that of the pivot by middle treemap, which has a lower value for the distance change metric (see previous paragraph).

The third row of Figure 8.10 shows that location drift of the Hilbert and Moore treemaps is lower than squarified, spiral, pivot by size and pivot by split size treemaps. Also, similar to the distance change metric, the location drift of Hilbert and Moore treemaps is higher than slice and dice and pivot by middle treemaps.

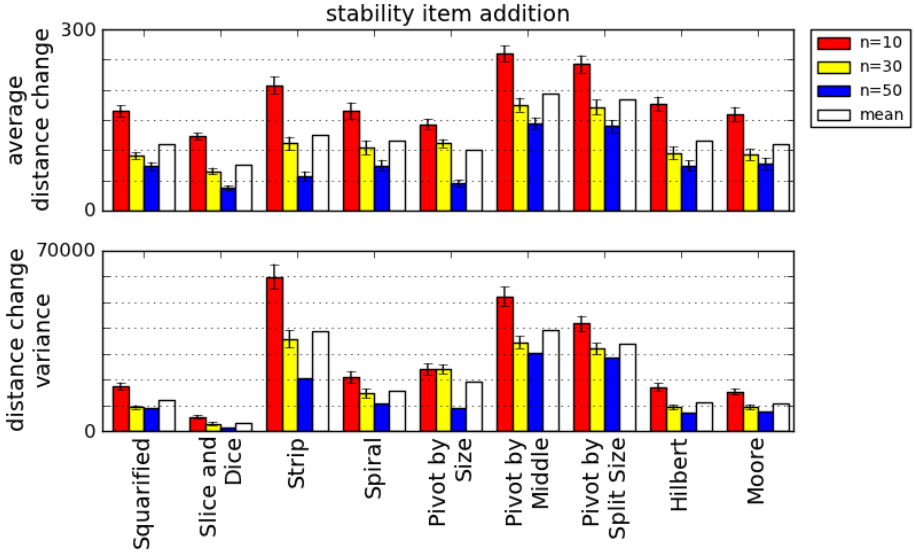


Figure 8.11: Distance change and distance change variance metrics for item addition. Error bars represent +/- 1 SE.

Stability - Item addition

Figure 8.11 shows the results of the comparison of different treemap layouts on the distance change metrics when an item is added to the layout.

The first row of Figure 8.11 shows that distance change of the Hilbert and Moore treemaps is lower than pivot by middle and by split size, but higher than slice and dice and pivot by size treemaps.

The second row of Figure 8.11 shows that distance change variance of the Hilbert and Moore treemaps is lower than most treemaps, but higher than slice and dice.

Most treemaps, including Hilbert and Moore, perform relatively similar in terms of stability when data is updated and when an item is added, but two layouts exhibit very different behaviours across these situations. The squarified treemap is quite unstable when the data is updated, but stability is not severely affected when an item is added. For the pivot by middle treemap, which performs very favourable in terms of stability when data updates, the reverse is true: it is very stable when the data is updated, but not when an item is added.

Stability score

In total, five stability metrics were calculated for each of the three set sizes (see Figures 8.10 and 8.11). To determine how well the different treemaps perform *overall* in terms of stability a 'stability score' was calculated by linearly normalising the stability metrics (scaling values between 0 and 1) and averaging them for each treemap. The stability scores for all treemaps are shown in Table 8.2.

Table 8.2: Stability scores and ranks for all treemaps. Lower scores mean better stability.

	Stability	
Treemap	Score	Rank
Squarified	0.59	7
Slice and Dice	0.00	1
Strip	0.63	8
Spiral	0.46	5
Pivot by Size	0.45	4
Pivot by Middle	0.56	6
Pivot by Split Size	0.96	9
Hilbert	0.38	3
Moore	0.35	2

Table 8.2 shows that the Hilbert and Moore treemaps have the best (i.e., lowest) stability scores after the slice and dice treemap. However, the slice and dice treemap is not an attractive option for 'real life' use because of the very high average aspect ratio (see next section), which makes items hard, or even impossible, to label and select.

Remaining metrics

Figure 8.12 shows the results of the comparison of different treemap layouts on the three other common treemap metrics: aspect ratio, continuity and readability.

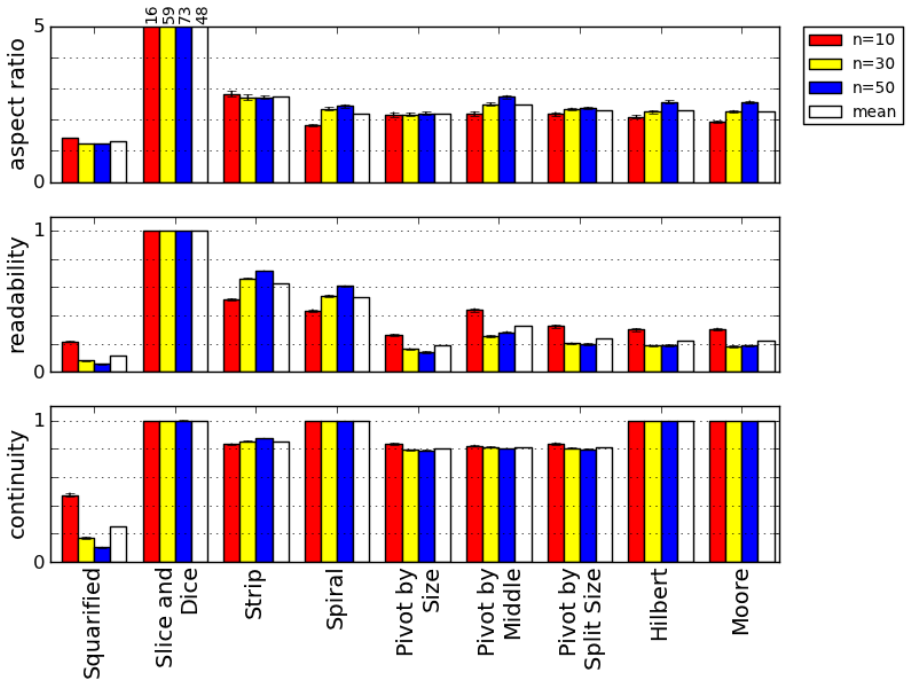


Figure 8.12: Aspect ratio, readability, and continuity metrics. Error bars represent +/- 1 SE.

The first row of Figure 8.12 shows that the Hilbert and Moore algorithms create treemaps with aspect ratios comparable to many other treemaps. Squarified treemaps perform particularly well (mean aspect ratios 1.4, 1.3 and 1.3 for the three set sizes, respectively, with 1 being the minimum) and slice and dice treemaps perform particularly poorly (mean aspect ratios 16, 59 and 73). All other algorithms produced mean aspect ratios in the range 1.8 to 2.8. Previous

work (Tu and Shen, 2007) also found that asides from the squarified and slice and dice treemaps, many treemaps are similar in terms of mean aspect ratio.

The second row of Figure 8.12 shows that the readability of the Hilbert and Moore treemaps is lower than slice and dice, strip, and spiral treemaps, but similar to pivot layouts. The poor readability of the Hilbert and Moore treemaps stems from the localised reorientation ('crumpling') of Hilbert and Moore curves. However, this poor performance on the readability metric need not be detrimental for user performance because the layout is very stable. Stable layouts, such as Hilbert and Moore treemaps, eliminate the need to perform such a (slow) linear search as encapsulated by the readability metric, because users learn where items are located. In other words, it is unlikely that the poor readability of the layout will affect user performance, because 'reading' the layout in order is not necessary to locate an item when the layout is very stable. Last, the third row of Figure 8.12 shows that Hilbert and Moore treemaps have maximal continuity.

Results summary

The results of the simulations show that the Hilbert and Moore treemaps have the best overall performance in terms of stability after the (very stable, but impractical due to its high aspect ratio) slice and dice treemap. This is explained by the good stability of the Hilbert and Moore treemaps both when data is updated and when an item is added to the layout. Combined with the low aspect ratio and good continuity of the treemaps generated by the Hilbert and Moore algorithms this suggests that Hilbert and Moore treemaps are very suitable to be used for the layout of application zones in SCOTZ.

8.7 Empirical study of location drift

Section 8.3 argued that the spatial stability of a treemap is not fully captured by the distance change metric, and introduced location drift to overcome this limitation. The previous section demonstrated that location drift can discriminate between designs (see Figure 8.10). However, for location drift to be a useful metric for the evaluation of treemap layouts, it should also be shown to relate to user performance, similar to the validation of the readability metric in Bederson et al. (2002). This section presents an empirical study which examined how various

levels of distance change and location drift affect user performance. In particular, it examines whether there is evidence that layouts with low distance change, but high location drift impair user performance, as proposed in Section 8.3.

The study used five different layout conditions with widely differing spatial properties resulting in substantially different metric values for distance change and location drift. The participants' tasks involved repeatedly selecting items within the layouts as quickly and accurately as possible. All layouts consisted of 36 items in a 6×6 matrix.

8.7.1 Layouts

The five experimental layout conditions included two control conditions representing end points of spatial stability: *random*, in which the location of every item was randomly assigned prior to each selection trial; and *stable*, in which item locations were fixed.

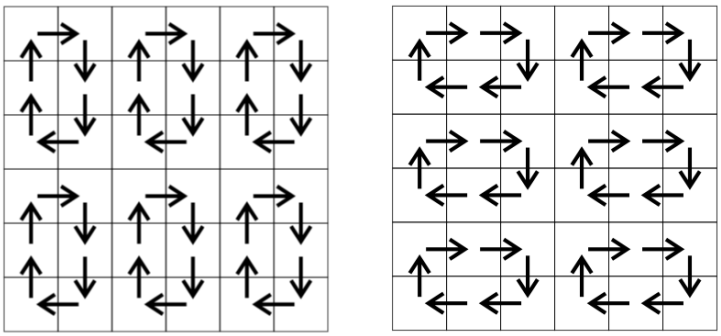


Figure 8.13: Layout updates for the *low distance change* condition.

The other three conditions were as follows:

Low distance change: Prior to each selection trial, item locations changed according to one of the 'rotation' methods shown in Figure 8.13 (randomly chosen).

Low location drift: All item locations within each quadrant (of nine items) in the layout were randomly assigned prior to each selection trial (i.e., all items always stay in the same quadrant, but within that quadrant their locations change).

Semi-random: All item locations were randomly assigned prior to each selection trial, with the caveat that 50% of items always stay in the same quadrant throughout all the selection trials.

Figure 8.14 shows the distance change and location drift metrics for each of these layouts. The random layout has a mean distance change of 525 pixels (approximately half the total layout size, which is 1024 by 1024 pixels) and a location drift of 36. For readability, the metrics in Figure 8.14 are normalised based on the metrics of the random layout ('max' denotes the the values for the the *random* layout).

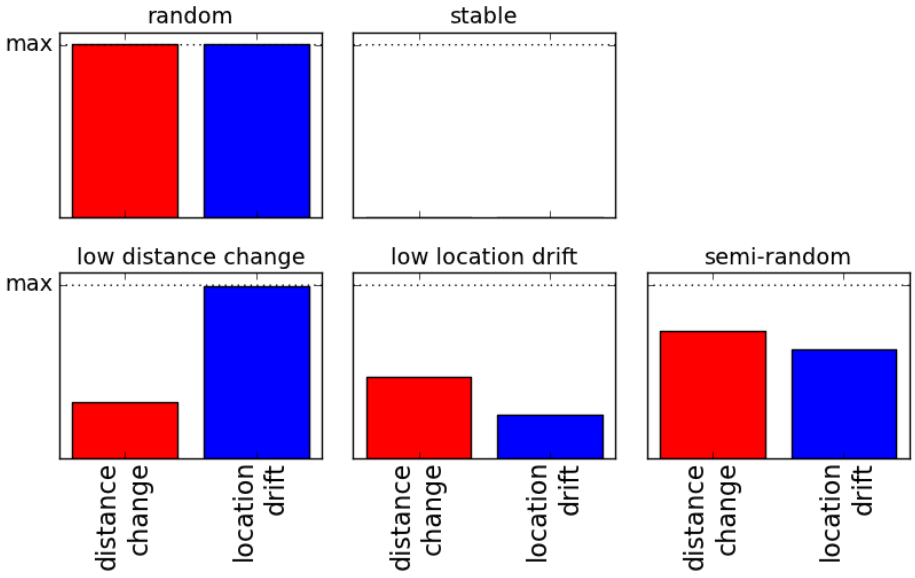


Figure 8.14: Metrics for the layouts used in the experiment.

Procedure

The experimental interface consisted of a 36-item square grid with distinct icons on the left side of the screen, and a cued target on the right side (see Figure 8.15). Participants were instructed to click on the target icon region as quickly and accurately as possible. After each selection, the grid was temporarily hidden from view and updated according to specifics of the layout (see previous section). Next, the participant was cued to press a 'next' button in the middle of the screen, after which a new item was cued, etc. A Zipfian distribution of targets was used: one item was cued 18 times, one 9 times, then (6, 5, 4, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1) for the others (α=1.0, R^2=.97). The items were cued in random order.

Figure 8.15: The experimental interface.

Design

The *selection time* dependent measure was analysed using a 5×3 RM-ANOVA for factors *layout* and *experience* (low, medium or high). Experience was determined by assigning first-time item selections as low experience, 2^{rd}–14^{th} selections as medium experience, and 15^{th}–18^{th} selections as high experience.

The experiment used a within-subject design, with all participants performing tasks in all layouts. The different layouts were presented in random order. After completion of the series of tasks in each layout the icons that had to be selected more than once in that layout were deleted from the icon collection, such that they were not re-used in following conditions. The experimental interface was displayed on a monitor with 1280 × 1024 pixels resolution.

8.7.2 Participants

Thirty students, naïve about the goal of the experiment, participated (15 male, 15 female, 18-39 years old). Participation lasted approximately 20 minutes.

8.7.3 Results and discussion

Any trial requiring more than one click to select the target was deemed an error and was removed from the analysis ($\sim$ 1% of selections).

There were significant main effects[1] for both *layout* ($F_{2.9,84.1}$=22, p<.001) and *experience* ($F_{1.4,39.5}$=133, p<.001). The *stable* layout was the fastest (1.4s), followed by the *low location drift* layout (1.7s), , the *semi-random* layout (1.9s), the *low distance change* layout (1.9s) and the *random* layout (1.9s). Post hoc comparisons (Bonferroni correction, α=.05) show pairwise differences between all layouts and the *stable* layout. Figure 8.16 shows a significant *layout*×*experience* interaction ($F_{4.2,122.9}$=13, p<.001), caused by relatively constant performance across experience with the *random*, *semi-random* and *low distance change* layouts in contrast to marked improvement of user performance with the *low location drift* and *stable* layouts.

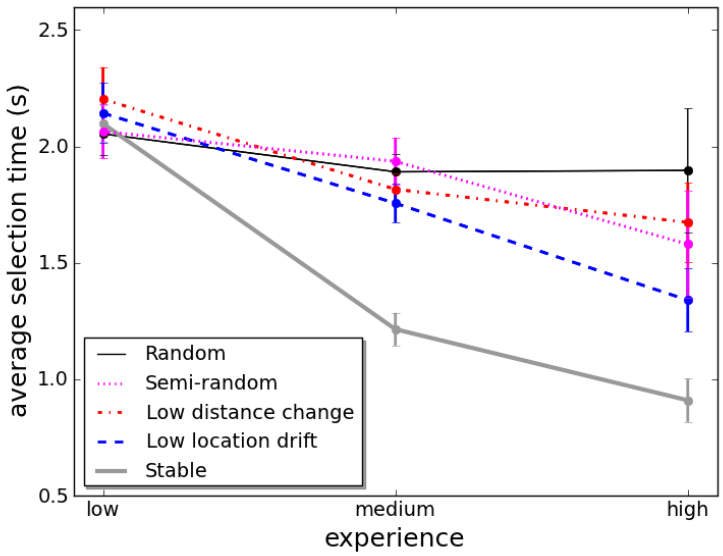


Figure 8.16: Experiment mean selection times for the five *layouts* by *experience* including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008).

The design of SCOTZ aims to maximise stability because it supports rapid acquisition of *familiar* targets in particular. To gain further insight into user performance on familiar targets, this section presents planned analysis of data from the 'high' experience level. The results show a significant main effect for *layout*

[1] Analysis corrected for deviance from sphericity by means of Greenhouse-Geisser where required.

($F_{2.6,75.1}$=15, p<.001). The *stable* layout was the fastest (0.9s), followed by the *low location drift* layout (1.3s), the *semi-random* layout (1.6s), the *low distance change* layout (1.7s), and the *random* layout (1.9s).

Post hoc analysis (Bonferroni correction, α=.05) reveals that the *stable* layout is faster than all other layouts (all p's <.001). Also, *low location drift* layout significantly improves performance compared to the *random* layout (p<.01) and the *low distance change* layout (p<.05).

The finding that the *low location drift* layout significantly improves performance compared to the *low distance change* layout is interesting, as the former is a layout with higher average distance change than the latter. However, location drift of the former is much lower, which provides a good explanation of why it performs better. When location drift is low the development of spatial memory for item locations is aided; it is easier to learn item locations when they stay in the same area of the screen.

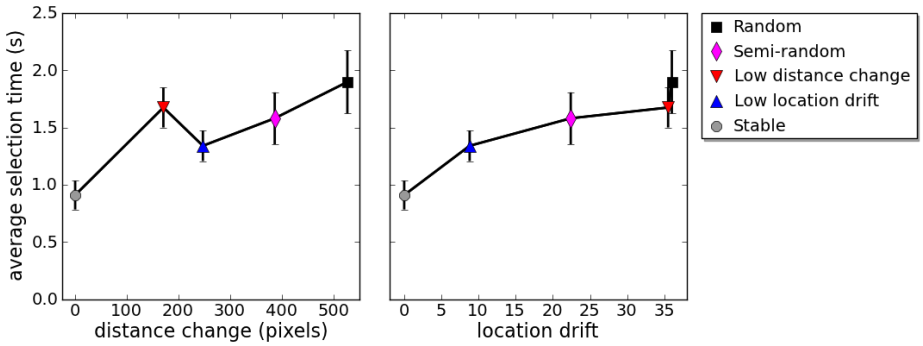


Figure 8.17: Experiment mean selection times on the 'high' experience level, split and sorted by distance change and location drift, including 95% within-subjects confidence intervals (Loftus and Masson, 1994; Cousineau, 2005; Morey, 2008). These plots are a combination of the data presented in Figure 8.14 and 8.16.

The results of the study suggest that location drift is a more robust predictor of user performance than distance change. This is shown in Figure 8.17. Figure 8.17 shows the average selection times on the 'high' experience level split and sorted by distance change (left) and location drift (right). The 'peak' in Figure 8.17 (left) is the *low distance change* layout, which has relatively low distance change, but high location drift. Correlation analysis of the data shown in Figure 8.17 reveals a significant strong correlation between location drift and selection time (r=.95,

$p<.05$), but the correlation between distance change and selection time fails to reach significance ($r=.85$, $p>.05$). On the individual (within subject) level (Bland and Altman, 1995) correlation coefficients for location drift and distance change (and time) are $r=.56$ ($p<.001$), and $r=.50$ ($p<.001$), respectively.

8.7.4 Results summary

The results show that (1) a layout with low distance change, but high location drift impairs user performance and (2) location drift seems to be a more robust predictor of user performance than distance change. This demonstrates that location drift is a valuable metric for the evaluation of treemaps and to accurately capture the stability of a treemap layout location drift needs to be taken into account

8.8 Conclusion

This chapter introduced two new treemaps, Hilbert and Moore. The Hilbert and Moore treemaps have very good stability, while aspect ratio is low and continuity is high.

The new location drift metric (1) is able to discriminate between different treemap layouts, (2) can explain some effects on user performance which the commonly used distance change metric can not account for and (3) appears to be a more robust predictor of user performance than the distance change metric. Additionally, the finding that the *low location drift* layout significantly improves performance compared to the *low distance change* layout is interesting, as the former is a layout with higher average distance change than the latter. However, location drift of the former is much lower, which provides a good explanation of why it performs better. When location drift is low development of spatial memory of item locations is aided; it is easier to learn item locations when they stay in the same area of the screen. These results show that to accurately capture the stability of a treemap layout not only distance change needs to be analysed, but location drift as well.

In all, Hilbert and Moore are good treemaps to be used in SCOTZ because of their good stability when data is updated, when many data updates have taken place and when items are added or deleted.

Chapter IX

Discussion and Future Work

9.1 Summary and research objectives

The research presented in this thesis addressed two research objectives: (1) to further explore and characterise how people switch between and organise windows and (2) to develop a new window switching interface that specifically tries to exploit the characteristics of window use, as well as provide a layout that is spatially stable. The first objective was reached with the longitudinal log study using *PyLogger*. Using data analysis and visualisations generated with *Window Watcher* tool many insights were gained about how people interact with windows, and how they make use of the various methods available to switch between windows. The strong revisitation patterns for application/window switching observed in the *PyLogger* study, combined with the desire to create a window switching tool which is spatially stable, led to the development of SCOTZ. Various empirical studies demonstrated the success of SCOTZ and its underlying principles. Finally, the various studies inspired the development of two new treemap algorithms, Hilbert and Moore, to meet the requirements of SCOTZ in terms of spatial stability.

9.2 Limitations of work

This section discusses various limitations of the work presented in this thesis: the user population of the studies, potential privacy concerns by the participants in the PyLogger study and the fact that most studies took place in a lab setting.

9.2.1 User study population

The participants in the studies reported in this thesis were predominantly students or employees of a university's computer science department. This implies that the

population is one of advanced/expert users. For some of the results of PyLogger study (Chapter 5) this limits generalisability. For example, usage patterns might differ for a less advanced user population. Therefore, future research could focus on different types of user groups. Nevertheless, the comparison with the results found in Hutchings et al. (2004) is valid, as this study had a very similar user population. Also, issues found regarding the (in)efficiency of different Taskbar button types and the somewhat limited use of the Alt+Tab functionality can be expected to be similar, if not exasperated, for a less advanced user population. Revisitation patterns are nearly identical to previous work and show little inter-subject variability, and are therefore not likely to be different for a less advanced user population.

In the various studies examining spatial stability and size morphing (Chapter 7) and the location drift study (Chapter 8) it is unlikely that the characteristics of the user population have affected the results, as these studies comprised of simple visual search and selection tasks, for which no effect of computer experience can be expected.

In the lab study comparing SCOTZ with the Windows 7 Taskbar and Alt+Tab (Chapter 7) the user population probably affected the results in the Alt+Tab condition. Most probably, a less advanced user population would have performed worse in the Alt+Tab condition, as the ordering of the windows in the Alt+Tab window is difficult to understand and the key combination requires some practice and keyboard dexterity.

9.2.2 Privacy concerns

Some participants in the PyLogger study might have had concerns about their privacy, as window titles were recorded, and therefore could have changed their behaviour because they knew they were being observed. However, many participants commented that they had completely forgotten about the logging tool when they were contacted again after the three-week period (note: participants were at this point reminded verbally that they could choose to delete any data before handing it over to the researchers). Also, a post-study questionnaire revealed that participants rarely consciously considered that the logger was running, and rarely to never changed their behaviour because the logger was running.

9.2.3 Lab setting

All studies (except the PyLogger study and qualitative study reported in Section 7.5) reported in this thesis took place in a lab setting. As the majority of studies, as noted before, comprised of simple visual search and selection tasks there is no reason to expect the results do not apply outside the lab setting. If anything, the lab setting minimised distraction. For the study which compared SCOTZ with the Windows 7 Taskbar and Alt+Tab the lab setting implied that the participants were using an unfamiliar computer, possibly with a different layout (or even different operating system) than the system they use normally. While this could mean that some participants were familiar with the Windows 7 Taskbar and Alt+Tab while others were not this cannot have unfairly advantaged SCOTZ in any way as (1) all participants were not familiar with SCOTZ and (2) the study used a within-subject design.

However, as all studies concerning SCOTZ took place in a lab setting (except the qualitative study reported in Section 7.5) it remains unknown how SCOTZ performs under 'real-life' conditions. Nevertheless, the various studies in Chapter 7 validate the underlying principles of SCOTZ, and there is no reason to expect these do not apply under more realistic conditions. A final question that remains unanswered is how to best introduce a tool (like SCOTZ) which provides a new mechanism for an action for which the fundamentals of the interaction have hardly changed for many years. After all, habits are hard to break (e.g. Singley and Anderson, 1989).

9.3 Extending the results

This section explores to what extent the results reported in this thesis can (and can *not*) be extended to other applications, research areas and contexts.

9.3.1 An empirical characterisation of window use

The results presented in Chapter 5 mainly focused on aspects related to window switching, but the results are informative for research involving other aspects of window use as well.

For example, several results of the empirical study suggest that people experience difficulty keeping track of the position of windows, both in terms of the *x*,*y*-position and the position in the *z*-order: (1) even when a direct click on the window to switch to it is possible people do not always use it and (2) analysis of the use of Alt+Tab reveals that this method is mainly used to switch back to the most recently used window, while going one step 'further' than that could potentially cover almost all window switches. These results suggest that users have trouble keeping track of windows. Users might benefit from more feedback about the location of windows on the screen, both in terms of *x*,*y*-position as well as the relative position of windows in the *z*-order.

A second example of the extendability of the results presented in Chapter 5 is related to the 'splattering' window management style. Better support for tiling, or viewing multiple window simultaneously with little overlap, might benefit users who prefer a splattering window management style. Microsoft Windows 7 does provide some support for tiling windows: when the user drags a window to the left/right edge of the screen it automatically expands to fill half of the screen. This allows for two windows to be tiled next to each other with relative ease. Including more features to facilitate displaying multiple windows (also see section 2.3.2) might support splatterers better in their window use and management.

9.3.2 Visualising large data sets

Chapter 4 presented *Window Watcher*, a tool for visualising the large data files generated by *PyLogger*. The goals of *Window Watcher* were to visualise (1) spatiotemporal, (2) spatial and (3) temporal data aspects, as well as (4) user actions. Spatiotemporal data is effectively captured by the 'playback' window, which provides a visual replay of the data sets, albeit sped up. The visualisation of this spatiotemporal aspect has proven the most useful in my analysis, not only because it reveals patterns (nearly) impossible to extract from the log data, it also inspired the analysis, in particular the improved definitions of window management styles. The spatial aspect of the data is captured by the heatmap, though I observed that due to the 'additive' effect of the heatmap (i.e., with every update the 'popularity' of the various areas of the screen is *cumulatively* recalculated) temporal patterns are visible as well when the heatmap is viewed while it is being dynamically

updated. These temporal aspects of the data have proven most difficult to capture effectively; when the visualisation is 'zoomed in' on too small a timeframe all context (previous/next behaviours) is lost, while a larger timeframe means it quickly becomes too coarse to observe small, but important details. Finally, the visualisation of user actions in *Window Watcher* is limited to mouse actions, and does not include keyboard actions. Though a 'replay' of Alt+Tab use, for example, could theoretically be implemented as well, this is a case where information is more effectively gathered from the log data (e.g., the average position of the target window in Alt+Tab).

9.3.3 Support for revisitation with treemaps

Section 6.4 already showed how support for revisitation with treemaps could be applied in a different context, i.e., website revisitation. Figure 9.1 shows another example of how size morphing could be applied in a different context: the (Windows) desktop environment. The Windows desktop contains (shortcuts to) applications and documents. By increasing the size of the icons a layout resembling an 'iconic tagcloud' is generated, and the most accessed applications become easier to find and select.

Figure 9.1: A mockup of a Windows desktop with size morphing applied to the application shortcut icons.

Another example of how the design principles presented in this thesis could be used in a different context is shown in Figure 9.2. Figure 9.2 shows the *7 Sticky Notes* application (`http://www.7stickynotes.com/`) for Microsoft Windows 7. With *7 Sticky Notes*, users can place 'post-it' notes on the desktop, and modify their colour, size, location, and so forth. Alternatively though, all notes could be equal sized to start off with, and be set to more/less important by the user, in response to which the notes will grow/shrink. If a Hilbert/Moore treemap is used, it means that the notes can dynamically grow/shrink, while the locations do not change much, and new notes can be added without affecting the stability of the layout much. This means that the user can focus his/her attention on the goal of the application (making notes and allocating importance to them), without having to spend (cognitive) effort on the spatial layout of the notes.

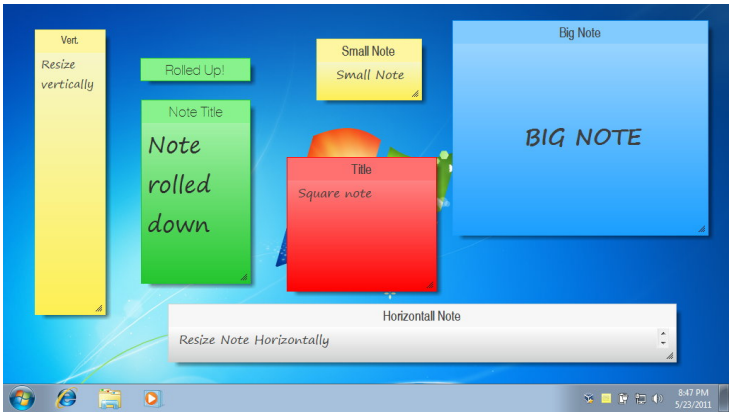


Figure 9.2: The *7 Sticky Notes* application (Source: `http://www.7stickynotes.com/screenshots.php`)

9.3.4 The importance of spatial stability

The experiment in Chapter 7 demonstrated the importance of spatial stability of a user interface. Many elements of the user interface on a 'standard' computer are already quite stable, e.g., the icons on the desktop do not move, and the order of the list of bookmarks in a web browser does not change. SCOTZ demonstrates how such stability can be achieved for transient elements such as windows. However, people who own or use several computers (e.g., at the workplace and at home) do not experience such stability across contexts unless they

invest (a lot) of effort into 'mirroring' both environments (note that SCOTZ's settings and variables are easily transportable between computers). Automatic synchronisation systems do exist for specific applications (e.g., `http://www.bookmarksync.com/`), but there is no easy method to mirror the layout of, for example, the desktop (including desktop icons, and applications that are 'pinned' to the Taskbar in Windows 7) across systems. Possibly, users who use several different computers regularly could benefit from improved support for synchronizing (desktop) layouts between these various environments, such that they have a spatially stable computing environment regardless of which computer they are using.

9.3.5 Keyboard users

Although SCOTZ provides support for keyboard users (by facilitating Alt+Tabbing through all the windows) it is primarily a mouse-driven system. Despite the fact that one 'heavy' Alt+Tab user in the qualitative study (see section 7.5) remarked that the size morphing in SCOTZ helped him to guide attention towards the application he was aiming for when using Alt+Tab, this seems to be a coping strategy rather than an actual *benefit* of SCOTZ. When using Alt+Tab to switch between windows with SCOTZ, the order in which it steps through the windows is consistent with Alt+Tab's default behaviour, but (unavoidably) rather confusing, as this does not match the order in which SCOTZ displays the windows. Future work should explore window switching tools for keyboard users further (for more details, see section 9.4.3).

9.3.6 SCOTZ on large and small displays

SCOTZ was mainly designed for, and tested on (see Chapter 7) 'regular' single or multiple monitor setups (when using a multiple monitor setup SCOTZ will appear on the screen where the mouse cursor was located when SCOTZ was invoked). This section explores to what extent SCOTZ could be successful on either (very) large or small displays.

Large displays

As noted in section 6.3.3, SCOTZ is suitable for large displays as it can be displayed as a small window positioned under the mouse cursor, thus preventing the user from having to move the mouse cursor over large distances. However, a general problem with window switching on large displays is that switching to a window will only cause it to be shifted to the top of the z-order, but its x,y-position is unaffected. SCOTZ's behaviour is consistent with other window switching interfaces on this aspect, and therefore behaves predictably. However, users of large displays (or even 'normal' multi-monitor setups) sometimes fail to notice where the focal window is located, as it may be outside the their field of view. Hoffmann et al. (2008) already demonstrated the success of various graphical cues (highlighting the target window and presenting graphical trails) to alleviate this problem. Though outside the scope of this thesis, an alternative approach could be to not only shift the focal window to the top of z-order, but also change its x,y-position to bring it closer to where the user is currently focusing. To prevent all windows ending up on one big 'heap' on one screen location, options for sending windows back to the periphery or outside the field of view should be implemented as well.

Small displays

Small-screen devices such as smartphones are often application-centric, with an application either running or not running (multiple entities of an application are uncommon or impossible), and applications usually taking up all available screen estate. This means that users are always reliant on a switching interface to switch between tasks/applications (as opposed to the use of a direct click on the target).

A default feature for switching between applications on the Android mobile OS is a list with the six most recently used applications, which is shown after a long press on the 'Home' button. Figure 9.3a shows a downloadable custom version of this functionality called MoreRecent (`http://code.google.com/p/morerecent/`), which shows the 15 most recently used applications, sorted by recency or application name.

Figure 9.3b shows how size morphing, similar to SCOTZ, could be applied in such an environment, with the most used applications growing in size. However,

(a) (b)

Figure 9.3: (a) The MoreRecent app for the Android OS (Source: `http://code.google.com/p/morerecent/`), (b) a mock-up of the MoreRecent app using size morphing.

care should be taken that the lesser used (and therefore smaller) applications do not become *too* small, which would make selecting them hard or impossible on a small screen.

9.4 Future work

This section explores several areas of future work that could lead to a better understanding of task and window switching, as well as address the needs of a specific group of users, i.e., keyboard shortcut users.

9.4.1 Improved understanding of user behaviour

While the results of the study presented in Chapter 5 revealed many interesting and novel findings regarding interaction with windows, many questions remain unanswered, in particular regarding the reasons for and the context of certain behaviour. For example, the finding that even when the target window is (almost) completely visible (90 to 100% visibility) another method rather than a direct click is used for 26.4% of window switches begs the question *why* this is happening.

Did the user not see the window and why? Was the user distracted? Was a window switching interface used purely out of habit? Unfortunately, such questions can not be answered by inspecting the log files, nor by using the visualisation methods presented in Chapter 4. Therefore, future work should focus on determining *why* certain behaviours happen, e.g., by think-aloud protocols, or by simulating certain 'critical' cases in a lab setting. At the same time, asking people about the reasons for their behaviour and underlying mental processes is notoriously risky, as demonstrated in the classic study by Nisbett and Wilson (1977), where passersby in a shop were presented with pairs of identical garments, e.g., stockings:

> "*Subjects were asked to say which article of clothing was the best quality and, when they announced a choice, were asked why they had chosen the article they had. There was a pronounced left-to-right position effect, such that the right-most object in the array was heavily over-chosen. For the stockings, the effect was quite large, with the right-most stockings being preferred over the left-most by a factor of almost four to one. When asked about the reasons for their choices, no subject ever mentioned spontaneously the position of the article in the array. And, when asked directly about a possible effect of the position of the article, virtually all subjects denied it, usually with a worried glance at the interviewer suggesting that they felt either that they had misunderstood the question or were dealing with a madman.*" (Nisbett and Wilson, 1977, pp. 243-244)

9.4.2 Task switching on small displays

Section 9.3.6 described an example of how SCOTZ could be implemented on small-screen devices, such as a smartphone. However, the comparative success of such an installment of SCOTZ could be better predicted if more were known about application and task switching on such small mobile devices. While there are studies regarding switching between tasks on a mobile device and different contexts (e.g., Tamminen et al., 2004), there is little known about task switching *on* the mobile- or smartphone. On one hand, it seems reasonable to suspect to find similarities to switching on the desktop computer, such as the revisitation patterns described in section 5.8.2 (a few applications are revisited a lot, but others are hardly ever revisited). On the other hand, differences can be expected, as

these devices are more application-centric than desktop computers, and some applications on smartphones do not even accommodate having multiple documents opened concurrently for one application. This different approach influences how users can (and will) use these devices.

In general, the "know thy user" admonition applies equally well to mobile contexts as it does to window switching on the desktop computer. As smartphones are rapidly becoming more popular this is a useful avenue for task switching research to explore.

9.4.3 Enhanced support for keyboard users

Keyboard shortcut users are a small, but significant portion of computer users. Additionally, it is widely agreed that keyboard shortcuts are a way for users to be more efficient as they progress to higher levels of expertise, without compromising the learnability of the user interface for beginning users (Nielsen, 1993; Lane et al., 2005; Dix et al., 2003). Multiple studies have demonstrated performance advantages of keyboards shortcuts over toolbar icons and traditional menu structures (Lane et al., 2005; McLoone et al., 2003; Odell et al., 2004). Future work should investigate how keyboard users can be supported better in their window switching, and there are several problems to be addressed.

The first problem when it comes to addressing the needs of keyboard users is the fact that it remains elusive what triggers a computer user to become a keyboard shortcut user. Previous work has found a weak correlation between the number of hours per week someone uses a computer and the use of keyboard shortcuts (Peres et al., 2004, 2005). Once again, to "know thy user" seems desirable before formulating any design recommendations. Unfortunately, the empirical study presented in this thesis failed to shed light on the matter.

Second, it is not clear how well people understand Alt+Tab's ordering of windows. Though informally, during one of my studies a participant made the comment that it "would be great if Alt+Tab could be used to access the most recently used window", which is in fact the essence of what Alt+Tab does. This demonstrates a clear misunderstanding of how Alt+Tab works. The way Alt+Tab is used seems to confirm this confusion: people mainly use it to just go back to the most recently used window, not traversing any further down the list of windows, even

though more than 80% of all target windows of a window switch are either on position 1 or 2 in the Alt+Tab list. Improving users' understanding of Alt+Tab's order might promote more efficient use of it, though I note that understanding the 'inner workings' of an interface is not necessarily required to be able to use it proficiently.

Bibliography

Accot, J. and S. Zhai (2002). More than dotting the i's — Foundations for crossing-based interfaces. In *Proc. of CHI '02*, pp. 73–80.

Adamczyk, P. D. and B. P. Bailey (2004). If not now, when? The effects of interruption at different moments within task execution. In *Proc. of CHI '04*, pp. 271–278. ACM.

Agarawala, A. and R. Balakrishnan (2006). Keepin' it real: Pushing the desktop metaphor with physics, piles and the pen. In *Proc of CHI '06*, pp. 1283–1292. ACM Press.

Ahlström, D., J. Großmann, S. Tak, and M. Hitz (2009). Exploring new window manipulation techniques. In *Proc. of OzCHI 2009*, pp. 177–183. ACM.

Alexander, J. and A. Cockburn (2008). An empirical characterisation of electronic document navigation. In *Proc. of GI 2008*, pp. 123–130. Canadian Information Processing Society.

Alexander, J., A. Cockburn, S. Fitchett, C. Gutwin, and S. Greenberg (2009). Revisiting read wear: analysis, design, and evaluation of a footprints scrollbar. In *Proc. of CHI '09*, pp. 1665–1674. ACM.

Alexander, J., A. Cockburn, and R. Lobb (2008). AppMonitor: A tool for recording user actions in unmodified windows applications. *Behavior Research Methods 40*, 413–421.

Asahi, T., D. Turo, and B. Shneiderman (1995). Visual decision-making: using treemaps for the analytic hierarchy process. In *Proc. of CHI '95*, pp. 405–406. ACM.

Atterer, R., M. Wnuk, and A. Schmidt (2006). Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction. In *Proc. of WWW '06*, pp. 203–212. ACM.

Bailey, B. P., J. A. Konstan, and J. V. Carlis (2001). The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In *Proc. of INTERACT '01*, pp. 593–601. IOS Press.

Balzer, M., O. Deussen, and C. Lewerentz (2005). Voronoi treemaps for the visualization of software metrics. In *Proc. of SoftVis '05*, pp. 165–172. ACM.

Bannon, L., A. Cypher, S. Greenspan, and M. L. Monty (1983). Evaluation and analysis of users' activity organization. In *Proc. of CHI '83*, pp. 54–57. ACM Press.

Bardram, J., J. Bunde-Pedersen, and M. Soegaard (2006). Support for activity-based computing in a personal computing operating system. In *Proc. of CHI '06*, pp. 211–220. ACM Press.

Barreau, D. and B. A. Nardi (1995). Finding and reminding: file organization from the desktop. *SIGCHI Bull. 27*, 39–43.

Baudisch, P. and C. Gutwin (2004). Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proc. of CHI '04*, pp. 367–374.

Beaudouin-Lafon, M. (2001). Novel interaction techniques for overlapping windows. In *Proc. of UIST '01*, pp. 153–154. ACM.

Beauvisage, T. (2009). Computer usage in daily life. In *Proc. of CHI '09*, pp. 575–584. ACM.

Bederson, B. B., B. Shneiderman, and M. Wattenberg (2002). Ordered and quantum treemaps: Making effective use of 2D space to display hierarchies. *ACM Trans. Graph. 21*(4), 833–854.

Bernstein, M., J. Shrager, and T. Winograd (2008). Taskposé: exploring fluid boundaries in an associative window visualization. In *Proc. of UIST '08*, pp. 231–234. ACM.

Bi, X. and R. Balakrishnan (2009). Comparing usage of a large high-resolution display to single or dual desktop displays for daily work. In *Proc. of CHI '09*, pp. 1005–1014. ACM.

Blanc-Brude, T. and D. L. Scapin (2007). What do people recall about their documents? Implications for desktop search tools. In *Proc. of IUI '07*, pp. 102–111. ACM.

Bland, J. and D. Altman (1995). Statistics notes: Calculating correlation coefficients with repeated observations: Part 1 - correlation within subjects. *British Medical Journal 310*(6977), 446.

Blom, J. (2000). Personalization: a taxonomy. In *Proc. of CHI EA '00*, pp. 313–314. ACM.

Bly, S. A. and J. K. Rosenberg (1986). A comparison of tiled and overlapping windows. In *Proc. of CHI '86*, pp. 101–106.

Bruls, M., K. Huizing, and J. v. Wijk (2000). Squarified treemaps. In *Proceedings of Joint Eurographics and IEEE*, pp. 33–42. IEEE Press.

Bunt, A., C. Conati, and J. McGrenere (2007). Supporting interface customization using a mixed-initiative approach. In *Proc. of IUI '07*, pp. 92–101. ACM.

Byrne, M. D. (1993). Using icons to find documents: simplicity is critical. In *Proc. of CHI '93*, pp. 446–453. ACM.

Byrne, M. D., J. R. Anderson, S. Douglass, and M. Matessa (1999). Eye tracking the visual search of click-down menus. In *Proc. of CHI '99*, pp. 402–409. ACM Press.

Card, S. K. and A. Henderson, Jr. (1987). A multiple, virtual-workspace interface to support user task switching. In *Proc. of CHI '87*, pp. 53–59. ACM.

Card, S. K., J. D. Mackinlay, and B. Shneiderman (1999). *Readings in Information Visualization: Using Vision to Think.* Morgan-Kaufmann.

Chapuis, O. and N. Roussel (2007). Copy-and-paste between overlapping windows. In *Proc. of CHI '07*, pp. 201–210. ACM.

Chi, E. H. (2002). Improving web usability through visualization. *IEEE Internet Computing 6*(2), 64–71.

Cockburn, A. and P. Brock (2006). Human on-line response to visual and motor target expansion. In *Proc. of GI 2006*, pp. 81–87. Canadian Human-Computer Communications Society.

Cockburn, A., S. Greenberg, S. Jones, B. McKenzie, and M. Moyle (2003). Improving web page revisitation: Analysis, design, and evaluation. *Information Technology and Society: Special Issue on Web Navigation 3*(1), 159–183.

Cockburn, A., C. Gutwin, and S. Greenberg (2007). A predictive model of menu performance. In *Proc. of CHI '07*, pp. 627–636. ACM Press.

Cockburn, A., P. O. Kristensson, J. Alexander, and S. Zhai (2007). Hard lessons: Effort-inducing interfaces benefit spatial learning. In *Proc. of CHI '07*, pp. 1571–1580. ACM Press.

Cockburn, A., B. McKenzie, and M. JasonSmith (2002). Pushing back: evaluating a new behaviour for the back and forward buttons in web browsers. *International Journal of Human-Computer Studies 57*(5), 397–414.

Cousineau, D. (2005). Confidence intervals in within-subject designs: A simpler solution to loftus and masson's method. *Tutorial in Quantitative Methods for Psychology 1*(1), 42–45.

Cugini, J. and J. Scholtz (1999). VISVIP: 3D visualization of paths through web sites. In *Proceedings Tenth International Workshop on Database and Expert Systems Applications*, pp. 259–263.

Cutrell, E., M. Czerwinski, and E. Horvitz (2000). Effects of instant messaging interruptions on computing tasks. In *Proc. of CHI EA '00*, pp. 99–100. ACM Press.

Czerwinski, M. and E. Horvitz (2002). Memory for daily computing events. In *Proc. of British HCI '02*, pp. 229–245.

Czerwinski, M., E. Horvitz, and S. Wilhite (2004). A diary study of task switching and interruptions. In *Proc. of CHI '04*, pp. 175–182. ACM Press.

Czerwinski, M., G. Robertson, B. Meyers, G. Smith, D. Robbins, and D. Tan (2006). Large display research overview. In *Proc. of CHI EA '06*, pp. 69–74. ACM.

Czerwinski, M., G. Smith, T. Regan, B. Meyers, G. G. Robertson, and G. Starkweather (2003). Towards characterizing the productivity benefits of very large displays. In *Proc. of INTERACT '03*, pp. 9–16. IOS Press.

Czerwinski, M., M. van Dantzich, G. Robertson, and H. Hoffman (1999). The contribution of thumbnail image, mouse-over text and spatial location memory to web page retrieval in 3D. In *Proc. of INTERACT '99*, pp. 163–170. IOS press.

de Chiara, R., U. Erra, and V. Scarano (2004). A visual adaptive interface to file systems. In *Proc. of AVI '04*, pp. 366–369. ACM.

Dix, A., J. Finlay, G. Abowd, and R. Beale (2003). *Human-Computer Interaction* (3rd ed.). Prentice Hall.

Dolan, R. and J. Matthews (1993). Maximizing the utility of customer product testing: Beta test design and management. *Journal of Product Innovation Management 10*(4), 318–330.

Dragicevic, P. (2004). Combining crossing-based and paper-based interaction paradigms for dragging and dropping between overlapping windows. In *Proc. of UIST '04*, pp. 193–196. ACM.

Dragunov, A. N., T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker (2005). Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *Proc. of IUI '05*, pp. 75–82. ACM.

Drewes, H. (2010). Only one fitts' law formula please! In *Proc. of CHI EA '10*, pp. 2813–2822. ACM.

Dubroy, P. and R. Balakrishnan (2010). A study of tabbed browsing among mozilla firefox users. In *Proc. of CHI '10*, pp. 673–682. ACM.

Dulberg, M. S., R. Amant, and L. Zettlemoyer (1999). An imprecise mouse gesture for the fast activation of controls. In *Proc. of INTERACT '99*, pp. 375–382.

Duncan, J. and G. Humphreys (1989). Visual search and stimulus similarity. *Psychological Review 96*(3), 433–458.

Ehret, B. (2002). Learning where to look: Location learning in graphical user interfaces. In *Proc. of CHI '02*, pp. 211–218. ACM Press.

Eick, Stephen, G. (2001). Visualizing online activity. *Commun. ACM 44*(8), 45–50.

Engdahl, B., M. Köksal, and G. Marsden (2005). Using treemaps to visualize threaded discussion forums on pdas. In *Proc. of CHI EA '05*, pp. 1355–1358. ACM.

Farris, J. S., K. S. Jones, and B. A. Anders (2001). Acquisition speed with targets on the edge of the screen: An application of Fitts' law to commonly used web browser controls. In *Proc. of the Human Factors and Ergonomics Society 45th Annual Meeting*, pp. 1205–1209.

Faure, G., O. Chapuis, and N. Roussel (2009). Power tools for copying and moving: useful stuff for your desktop. In *Proc. of CHI '09*, pp. 1675–1678. ACM.

Finch, S. (2003). *Mathematical Constants*. Cambridge University Press, Cambridge UK.

Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology 47*, 381–391.

Fono, D. and R. Vertegaal (2005). EyeWindows: Evaluation of eye-controlled zooming windows for focus selection. In *Proc. of CHI '05*, pp. 151–160. ACM Press.

Frantz, T. L. and K. M. Carley (2005). Treemaps as a tool for social network analysis. Technical report, Center for Computational Analysis of Social and Organizational Systems.

Gaylin, K. B. (1986). How are windows used? Some notes on creating an empirically-based windowing benchmark task. In *Proc. of CHI '86*, pp. 96–100.

Goldberg, H. and A. Wichansky (2003). Eye tracking in usability evaluation: A practitioner's guide. In J. Hyn, R. Radach, and H. Deubel (Eds.), *The mind's eye: Cognitive and applied aspects of eye movement research*, pp. 493–516. Elsevier.

Goldberg, J. and X. Kotval (1999). Computer interface evaluation using eye movements: methods and constructs. *International Journal of Industrial Ergonomics 24*(6), 631–645.

González, V. M. and G. Mark (2004). "Constant, constant, multi-tasking craziness": Managing multiple working spheres. In *Proc. of CHI '04*, pp. 113–120. ACM.

Gray, M., A. Badre, and M. Guzdial (1996). Visualizing usability log data. In *Proc. of INFOVIS '96*, pp. 93–98. IEEE Computer Society.

Greenberg, S. and I. Witten (1993). Supporting command reuse: Empirical foundations and principles. *International Journal of Man-Machine Studies 39*, 353–390.

Grudin, J. (2001). Partitioning digital worlds: Focal and peripheral awareness in multiple monitor use. In *Proc. of CHI '01*, pp. 458–465.

Halverson, T. and A. J. Hornof (2008). The effects of semantic grouping on visual search. In *Proc. of CHI EA '08*, pp. 3471–3476. ACM.

Hansen, W. (1971). User engineering principles for interactive systems. In *Proc. of AFIPS '71*, pp. 523–532. ACM.

Hansen, W. J., D. R. Barstow, and Shrobe (1984). In *Interactive Programming Environments*, pp. 288–299. McGraw-Hill.

Haraty, M., S. Nobarany, S. DiPaola, and B. Fisher (2009). Adwil: adaptive windows layout manager. In *Proc. of CHI EA '09*, pp. 4177–4182. ACM.

Henderson, D. A. and S. Card (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.

Hilbert, D. (1891). Über die stetige abbildung einer line auf ein flächenstück. *Mathematische Annalen 38*, 459–460.

Hilbert, D. and D. Redmiles (1999). Extracting usability information from user interface events. *ACM Computing Surveys 32*, 384–421.

Hoffmann, R., P. Baudisch, and D. S. Weld (2008). Evaluating visual cues for window switching on large screens. In *Proc. of CHI '08*, pp. 929–938. ACM.

Holzinger, A. (2005). Usability engineering methods for software developers. *Commun. ACM 48*, 71–74.

Hong, J. and J. Landay (2001). Webquilt: a framework for capturing and visualizing the web experience. In *Proc. of WWW '01*. ACM.

Hornof, A. J. (2001). Visual search and mouse-pointing in labeled versus unlabeled two-dimensional visual hierarchies. *ACM Trans. Comput.-Hum. Interact. 8*, 171–197.

Hutchings, D., G. Smith, B. Meyers, M. Czerwinski, and G. Robertson (2004). Display space usage and window management operation comparisons between single monitor and multiple monitor users. In *Proc. of AVI'04*, pp. 32–39. ACM Press.

Hutchings, D. and J. Stasko (2007). Quantifying the performance effect of window snipping in multiple-monitor environments. In *Proc. of INTERACT '07*, pp. 461–474. Springer Berlin / Heidelberg.

Hutchings, D. R. and J. Stasko (2002). Quickspace: new operations for the desktop metaphor. In *Proc. of CHI EA '02*, pp. 802–803. ACM.

Hutchings, D. R. and J. Stasko (2004a). Revisiting display space management: understanding current practice to inform next-generation design. In *Proc. of GI '04*, pp. 127–134. Canadian Human-Computer Communications Society.

Hutchings, D. R. and J. Stasko (2004b). Shrinking window operations for expanding display space. In *Proc. of AVI '04*, pp. 350–353. ACM.

Iqbal, S. T. and E. Horvitz (2007). Disruption and recovery of computing tasks: Field study, analysis, and directions. In *Proc. of CHI '07*, pp. 677–686. ACM Press.

Ivory, M. Y. and M. A. Hearst (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv. 33*, 470–516.

Jacob, R. J. K. and K. S. Karn (2003). Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. In J. Hyönä, R. Radach, and H. Deubel (Eds.), *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*. Elsevier Science.

Johnson, B. and B. Shneiderman (1991). Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. of VIS '91*, pp. 284–291. IEEE Computer Society Press.

Kaasten, S. and S. Greenberg (2001). Integrating back, history and bookmarks in web browsers. In *Proc. of CHI EA '01*, pp. 379–380.

Kabbash, P., I. S. MacKenzie, and W. Buxton (1993). Human performance using computer input devices in the preferred and non-preferred hands. In *Proc. of CHI '93*, pp. 474–481.

Kamba, T., S. A. Elson, T. Harpold, T. Stamper, and P. Sukaviriya (1996). Using small screen space more efficiently. In *Proc. of CHI '96*, pp. 383–390. ACM.

Kandogan, E. and B. Shneiderman (1996). Elastic windows: Improved spatial layout and rapid multiple window operations. In *Proc. of AVI'96*, pp. 29–38. ACM Press.

Kandogan, E. and B. Shneiderman (1997). Elastic windows: evaluation of multi-window operations. In *Proc. of CHI '97*, pp. 250–257. ACM.

Kang, Y. and J. Stasko (2008). Lightweight task/application performance using single versus multiple monitors: a comparative study. In *Proc. of GI 2008*, pp. 17–24. Canadian Information Processing Society.

Kaptelinin, V. (2003). Umea: translating interaction histories into project contexts. In *Proc. of CHI '03*, pp. 353–360. ACM.

Kärkkäinen, L. and J. Laarni (2002). Designing for small display screens. In *Proc. of NordiCHI '02*, pp. 227–230. ACM.

Keim, D. A. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics 8*(1), 1–8.

Kellar, M., C. Watters, and M. Shepherd (2006). The impact of task on the usage of web browser navigation mechanisms. In *Proc. of GI 2006*, pp. 235–242.

Kumar, M., A. Paepcke, and T. Winograd (2007). EyeExposé: Switching applications with your eyes. Technical report.

Kumar, M. and T. Winograd (2007). Guide: gaze-enhanced ui design. In *Proc. of CHI EA '07*, pp. 1977–1982. ACM.

Lane, D. M., H. A. Napier, S. C. Peres, and A. Sandor (2005). Hidden costs of graphical user interfaces: Failure to make the transition from menus and icon toolbars to keyboard shortcuts. *International Journal of Human Computer Interaction 18*(2), 133–144.

Lee, E. and J. McGregor (1985). Minimizing user search time in menu retrieval systems. *Human Factors 27*(2), 157–162.

Lewis, J. P., R. Rosenholtz, N. Fong, and U. Neumann (2004). VisualIDs: automatic distinctive icons for desktop interfaces. *ACM Trans. Graph. 23*, 416–423.

Loftus, G. and M. Masson (1994). Using confidence intervals in within-subject designs. *Psychonomic Bulletin & Review 1*, 476–490.

Mackay, W. (1991). Triggers and barriers to customizing software. In *Proc. of CHI '91*, pp. 153–160. ACM.

MacKenzie, I. S. and W. Buxton (1992). Extending fitts' law to two-dimensional tasks. In *Proc. of CHI '92*, pp. 219–226.

MacKenzie, I. S., A. Sellen, and W. Buxton (1991). In *Proc. of CHI '91*, pp. 161–166.

Mackinlay, J. D. and C. Royer (2004). Log-based longitudinal study finds window thrashing. Technical report.

MacLean, A., K. Carter, L. Lovstrand, and T. Moran (1990). In *Proc. of CHI '90*, pp. 175–182.

Malone, T. W. (1983). How do people organise their desks? Implications for the design of office information systems. *ACM Trans. on Office Information Systems 1*(1), 99–112.

Mark, G., V. M. Gonzalez, and J. Harris (2005). No task left behind? Examining the nature of fragmented work. In *Proc. of CHI '05*, pp. 321–330. ACM Press.

McGuffin, M. and R. Balakrishnan (2002). Acquisition of expanding targets. In *Proc. of CHI '02*, pp. 57–64. ACM Press.

McLoone, H., K. Hinckley, and E. Cutrell (2003). Bimanual interaction on the microsoft office keyboard. In *Proc. of INTERACT 2003*. IOS Press.

Milic-Frayling, N., R. Sommerer, and K. Rodden (2003). WebScout: support for revisitation of web pages within a navigation session. In *Proc. of WI '03*, pp. 689–693.

Moore, E. H. (1900). On certain crinkly curves. *Transactions of the American Mathematical Society 1*, 72–90.

Morey, R. (2008). Confidence intervals from normalized data: A correction to cousineau (2005). *Tutorial in Quantitative Methods for Psychology 4*(2), 61–64.

Moyes, J. (1994). When users do and don't rely on icon shape. In *Proc. of CHI '94*, pp. 283–284.

Myers, B. A. (1988). A taxonomy of window manager user interfaces. *IEEE Computer Graphics and Applications 8*(5), 65–84.

Neisser, U., R. Novick, and R. Lazar (1964). Searching for novel targets. *Perceptual and Motor Skills 19*, 427–432.

Niedermeier, R., K. Reinhardt, and P. Sanders (1997). Towards optimal locality in mesh-indexings. In *Proceedings of the 11th International Symposium on Fundamentals of Computation Theory*, pp. 364–375. Springer-Verlag.

Nielsen, J. (1993). *Usability Engineering* (1st ed.). Morgan Kaufmann.

Niemela, M. and P. Saariluoma (2003). Layout attributes and recall. *Behavior and Information Technology 22*(5), 353–363.

Nisbett, R. and T. Wilson (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review 84*(3), 231–259.

Obendorf, H., H. Weinreich, E. Herder, and M. Mayer (2007). Web page revisitation revisited: implications of a long-term click-stream study of browser usage. In *Proc. of CHI '07*, pp. 597–606. ACM.

Odell, D. L., R. C. Davis, A. Smith, and P. K. Wright (2004). Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques. In *Proc. of GI 2004*, pp. 17–24. Canadian Human-Computer Communications Society.

Oliver, N., M. Czerwinski, G. Smith, and K. Roomp (2008). RelAltTab: assisting users in switching windows. In *Proc. of IUI '08*, pp. 385–388. ACM.

Oliver, N., G. Smith, C. Thakkar, and A. Surendran (2006). Swish: Semantic analysis of window titles and switching history. In *Proc. of IUI '06*, pp. 194–201. ACM Press.

Oulasvirta, A. and P. Saariluoma (2006). Surviving task interruptions: Investigating the implications of long-term working memory theory. *International Journal of Human-Computer Studies 64*(10), 941–961.

Parasuraman, R. (1986). Vigilance, monitoring, and search. In K. Boff, L. Kaufman, and J. Thomas (Eds.), *Handbook of perception and human performance. Volume 2*. Wiley.

Peano, G. (1890). Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen 36*, 157–160.

Perer, A. and B. Shneiderman (2008). Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In *Proc. of CHI '08*, pp. 265–274. ACM.

Peres, S. C., M. D. Fleetwood, F. P. Tamborello II, M. Yang, and D. L. Paige-Smith (2005). Pros, cons, and changing behavior: An application in using

the keyboard to issue commands. In *Proc. of Human Factors and Ergonomics Society 49th Annual Meeting*, pp. 637–641.

Peres, S. C., F. P. Tamborello II, M. D. Fleetwood, P. Chung, and D. L. Paige-Smith (2004). Keyboard shortcut usage: The roles of social factors and computer experience. In *Proc. of Human Factors and Ergonomics Society 48th Annual Meeting*, pp. 803–807.

Pettigrew, A. (1990). Longitudinal field research on change: Theory and practice. *Organization Science 1*(3), 267–292.

Plaisant, C. (2004). The challenge of information visualization evaluation. In *Proc. of AVI '04*, pp. 109–116. ACM.

Poole, A. and L. Ball (2005). Eye tracking in human-computer interaction and usability research. In C. Ghaoui (Ed.), *Encyclopedia of human computer interaction*, pp. 211–219. Idea group.

Preece, J., Y. Rogers, and H. Sharp (2007). *Interaction Design: Beyond human-computer interaction* (2nd ed.). John Wiley & Sons.

Ramos, G., G. Robertson, M. Czerwinski, D. Tan, P. Baudisch, K. Hinckley, and M. Agrawala (2006). Tumble! Splat! Helping users access and manipulate occluded content in 2D drawings. In *Proc. of AVI '06*, pp. 428–435. ACM.

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

Robertson, G., M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich (1998). Data mountain: Using spatial memory for document management. In *Proc. of UIST'98*, pp. 153–162.

Robertson, G., E. Horvitz, M. Czerwinski, P. Baudisch, D. Hutchings, B. Meyers, D. Robbins, and G. Smith (2004). Scalable fabric: Flexible task management. In *Proc. of AVI'04*, pp. 85–89. ACM Press.

Robertson, G., M. van Dantzich, M. Czerwinski, K. Hinckley, D. Thiel, D. Robbins, K. Risden, and V. Gorokhovsky (2000). The task gallery: A 3D window manager. In *Proc. of CHI '00*, pp. 494–501.

Roediger, H. L. (1990). Implicit memory: Retention without remembering. *American Psychologist 45*(9), 1043–1056.

Sagan, H. (1994). *Space-Filling Curves*. Springer.

Salvucci, D. and J. Goldberg (2000). Identifying fixations and saccades in eye-tracking protocols. In *Proc. of ETRA '00*, pp. 71–78. ACM.

Sauro, J. and J. R. Lewis (2010). Average task times in usability tests: what to report? In *Proc. of CHI '10*, pp. 2347–2350. ACM.

Schwarz, N. (1999). Self-reports: How the questions shape the answers. *American Psychologist 54*(2), 93–105.

Shi, K., P. Irani, and B. Li (2005). An evaluation of content browsing techniques for hierarchical space-filling visualizations. In *Proc. of InfoVis '05*, pp. 81–88. IEEE Computer Society.

Shneiderman, B. (1992). Tree visualization with tree-maps: 2-D space-filling approach. *ACM Trans. Graph. 11*(1), 92–99.

Shneiderman, B. (1997). Direct manipulation for comprehensible, predictable and controllable user interfaces. In *Proc. of IUI '97*, pp. 33–39. ACM.

Shneiderman, B. (2002). Inventing discovery tools: combining information visualization with data mining. *Information Visualization 1*, 5–12.

Shneiderman, B. and M. Wattenberg (2001). Ordered treemap layouts. In *Proc. of InfoVis '01*, pp. 73–78.

Singley, M. and J. Anderson (1989). *The transfer of cognitive skill*. Harvard University Press.

Smith, G., P. Baudisch, G. Robertson, M. Czerwinski, B. Meyers, D. Robbins, E. Horvitz, and D. Andrews (2003). Groupbar: The taskbar evolved. In *Proc. of OzCHI '03*, pp. 34–43.

Snowden, R., P. Thompson, and T. Troscianko (2006). *Basic Vision: An Introduction to Visual Perception*. Oxford University Press.

Stasko, J. (2000). An evaluation of space-filling information visualizations for depicting hierarchical structures. *Int. J. Hum.-Comput. Stud. 53*(5), 663–694.

Stuerzlinger, W., O. Chapuis, D. Phillips, and N. Roussel (2006). User interface façades: towards fully adaptable user interfaces. In *Proc. of UIST '06*, pp. 309–318. ACM.

Tamminen, S., A. Oulasvirta, K. Toiskallio, and A. Kankainen (2004). Understanding mobile contexts. *Personal Ubiquitous Comput. 8*, 135–143.

Tan, D. S., B. Meyers, and M. Czerwinski (2004). Wincuts: manipulating arbitrary window regions for more effective use of screen space. In *Proc. of CHI EA '04*, pp. 1525–1528. ACM.

Tashman, C. (2006). Windowscape: A task oriented window manager. In *Proc. of UIST '06*, pp. 77–80. ACM Press.

Tauscher, L. and S. Greenberg (1997). Revisitation patterns in world wide web navigation. In *Proc. of CHI '97*, pp. 399–406.

Teitelbaum, R. C. and R. E. Granda (1983). The effects of positional constancy on searching menus for information. In *Proc. of CHI '83*, pp. 150–153.

Tourangeau, R. (2001). Remembering what happened: Memory errors and survey reports. In A. Stone, J. Turkkan, C. Bachrach, J. Jobe, H. Kurtman, and V. Cain (Eds.), *The Science of Self-Report: Implications for Research and Practice*, pp. 29–47. Lawrence Erlbaum.

Tractinsky, N. and D. Zmiri (2005). Exploring attributes of skins as potential antecedents of emotion in HCIs. In P. Fishwick (Ed.), *Aesthetic Computing*. MIT Press.

Treisman, A. (1986). Properties, parts, and objects. In K. Boff, L. Kaufman, and J. Thomas (Eds.), *Handbook of perception and human performance*. Wiley.

Treisman, A. M. and G. Gelade (1980). A feature-integration theory of attention. *Cognitive Psychology 12*(1), 97–136.

Truemper, J. M., H. Sheng, M. G. Hilgers, R. H. Hall, M. Kalliny, and B. Tandon (2008). Usability in multiple monitor displays. *SIGMIS Database 39*, 74–89.

Tu, Y. and H. Shen (2007). Visualizing changes of hierarchical data using treemaps. In *IEEE Transactions on Visualization and Computer Graphics*, pp. 1286–1293.

Uehling, D. L. and K. Wolf (1995). User action graphing effort (UsAGE). In *Proc. of CHI '95*, pp. 290–291. ACM.

van Dam, A. (1997). Post-WIMP user interfaces. *Commun. ACM 40*, 63–67.

van Wijk, J. J. and H. van de Wetering (1999). Cushion treemaps: Visualization of hierarchical information. In *Proc. of INFOVIS '99*, pp. 73–78. IEEE Computer Society.

Wattenberg, M. (1999). Visualizing the stock market. In *Proc. of CHI EA '99*, pp. 188–189. ACM.

Wattenberg, M. (2005). A note on space-filling visualizations and space-filling curves. In *Proc. of InfoVis '05*, pp. 24–29. IEEE Computer Society.

Wickens, C. (1992). *Engineering psychology and human performance* (2nd ed.). HarperCollins.

Wickens, C., S. Gordon, and Y. Liu (1998). *An introduction to human factors engineering* (2nd ed.). Addison-Wesley.

Wolfe, J. and W. Gray (2007). Guided search 4.0: Current progress with a model of visual search. In W. Gray (Ed.), *Integrated Models of Cognitive Systems*, pp. 99–119. Oxford.

Wolfe, J. M., K. R. Cave, and S. L. Franzel (1989). Guided search: an alternative to the feature integration model for visual search. *Journal of Experimental Psychology: Human Perception and Performance 15*(3), 419–33.

Wolfe, J. M. and T. S. Horowitz (2004). What attributes guide the deployment of visual attention and how do they do it? *Nature Reviews Neuroscience 5*(6), 495–501.

Xu, Q. and G. Casiez (2010). Push-and-pull switching: window switching based on window overlapping. In *Proc. of CHI '10*, pp. 1335–1338. ACM.