Window Watcher: A Visualisation Tool for Understanding Windowing Activities

Susanne Tak and Andy Cockburn
University of Canterbury
Computer Science and Software Engineering,
Private Bag 4800, Christchurch 8140, New Zealand
susanne.tak@pg.canterbury.ac.nz, andy@cosc.canterbury.ac.nz

ABSTRACT
Almost all actions on a computer are mediated by windows, yet we know surprisingly little about how people coordinate their activities using these windows. Studies of window use are difficult for two reasons: gathering longitudinal data is problematic and it is unclear how to extract meaningful characterisations from the data. In this paper, we present a visualisation tool called *Window Watcher* that helps researchers understand and interpret low level event logs of window switching activities generated by our tool *PyLogger*. We describe its design objectives and demonstrate ways that it summarises and elucidates window use.

Author Keywords
Data visualisation, window switching, window management, longitudinal analysis

ACM Classification Keywords
H5.2 [User Interfaces]: Interaction styles, screen design.

INTRODUCTION
All activities on desktop computers are mediated through windows, which allow users to coordinate their work across space and time: for example, a user might allocate a particular screen region to a specific application; and users switch between windows to direct their attention to salient events, such as incoming email. The nature of windowing interfaces has remained relatively unchanged since first demonstrated in the 1973 Xerox Alto, but *many* researchers and developers have investigated methods to improve performance with them (e.g. Bernstein et al., 2008; Robertson et al., 2004; Smith et al., 2003). Their status as an elemental component of the desktop user interface means that any small improvement in interaction with windows will yield substantial gains when multiplied across hundreds or thousands of daily actions.

"Know thy user" is a widely touted incantation for successful interface design. Knowing what users want to achieve and how they achieve it is a fundamental requirement for iterative interface refinement. Yet we know surprisingly little about how users coordinate their work across windows – this knowledge gap limits our ability to redesign and improve windowing interfaces.

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
OZCHI '09, Nov 23-27, 2009, Melbourne, Australia
© ACM 2009 ISBN: 978-1-60558-854-4/09/11...$10.00

The knowledge gap exists for two main reasons. First, the following factors make it hard to gather data that might contribute to a characterisation of window switching activities. Longitudinal analysis is necessary 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. 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. Finally, direct observation methodologies are impractical for large scale longitudinal analyses, so log based automatic recording of user activities is preferable. Unfortunately, developing such software has, until recently, been prohibitively complex (Alexander and Cockburn, 2008; Kellar et al., 2006). Together, these factors mean it is difficult to *gather* data that might usefully characterise window switching.

The second reason contributing to the knowledge gap of window switching, and the focus of this paper, is that even when a large corpus of data that precisely describes user actions is available, it is very difficult to extract a meaningful characterisation. Pirolli *et al.* (1996) used the term 'silk from a sows ear' to capture the complexity of extracting valid and useful findings from low level raw data. Although Pirolli *et al.* were studying web site use rather than window interaction, their comment applies equally well to this domain.

We recently completed a log based longitudinal study (26 participants) of window switching actions that generated 495 MB of low-level logs. To help interpret this data we designed and constructed *Window Watcher,* a dynamic query visualisation tool, described in this paper. The tool affords a variety of valuable insights into the spatiotemporal aspects of window switching. The tool can replay abstract representations of real time or time-compressed window actions, it can summarise which applications are used on what regions of the screen, show the depth and number of active windows, and illuminate a variety of behaviours via dynamic filtering of the dataset. Visualisations of interactions with windows and/or the screen are very intuitive and powerful, as can be seen, for example, in the visual depiction of mouse event locations in Bi and Balakrishnan (2009).

In this paper we describe our visualisation tool and its potential as a research and development tool. Several real-world examples of window use data are used to demonstrate the capabilities of the tool. Also, we briefly describe the tool we used to collect window use data.

RELATED WORK

Two areas of related work are important for this study: studies of window use and the use of visualisation tools in other areas.

Window use

Multiple previous studies have found that users normally have many windows open, e.g. eight or more windows open 78.1% of the time (Hutchings et al., 2004). It has also been reported that the average number of simultaneously opened windows increases with available display space: from four for single monitor users to up to 18 for users with multiple monitors (Smith et al., 2003). As large monitors and multi-monitor setups are becoming increasingly popular, this implies that the number of open windows will also increase. This makes studies of window use of great interest to developers of tools for coping with this large number of open windows. In this section, we divide window use in two different categories: window switching and window management.

Window switching

Empirical studies of window switching behaviour all point in the same direction; window switching is a very common activity. For example, in a three week longitudinal study, Hutchings *et al.* (2004) found a mean window activation time of 20.9 seconds, and a median of only 3.77 seconds. This high frequency of window switching is confirmed by Mackinlay and Royer (2004), who also conducted a log analysis of window switching. Windows switching activities are amongst the most frequent windowing behaviour. Gaylin (1986) shows that window switching activities are far more frequent than window creation, deletion, or geometry management.

Studies of which methods people use to switch between windows are scarce. Kumar et al. (2007) categorise the standard tools for switching between windows into three categories according to how the tools order the selectable documents and how the selections are made: Temporal, Spatial and Hybrid. However, the use frequency of these tools remains largely unclear. Studies of window switching tools often focus on their relative efficiency and effectiveness. For example, in an empirical evaluation Microsoft Windows' Alt+Tab was found to be faster than Mac OS X's Exposé when the number of windows is small, but its performance decreased as the number of windows increased (Kumar et al., 2007). Also, when comparing the Windows Taskbar to Exposé, it was found that more errors are made with the Windows Taskbar, possibly due to the smaller target size compared to Exposé.

Though these findings are valuable, they do not capture how and how much window switching tools are used by actual users on a daily basis. Also, common user problems when switching between windows are still largely unknown.

Window management

How people organise windows on the screen is referred to as *window management*. Studies of window management techniques often differentiate between overlapping and tiling techniques (Bly and Rosenberg, 1986; Myers, 1988). For example, Bly and Rosenberg (1986) suggest that the optimal window management technique depends on the type of task at hand. If a task requires little window manipulation, tiling is preferable, if a task requires more window manipulation, overlapping windows are more suitable. Also, they suggest a possible effect of expertise; someone who is "inexpert" with using overlapping window might be better off using tiling in all cases. A more recent study has found that people hardly ever tile their windows, while the increasing screen size and/or number of monitors people does facilitate this (Hutchings and Stasko, 2004).

In a longitudinal study, Hutchings and Stasko (2004) observed three window management styles; *maximizers*, *near maximizers* and *careful coordinators*. Maximizers always maximize every window. Near maximizers resize (nearly) all windows to slightly smaller than the desktop size, leaving some space open for certain desktop icons or smaller windows. Careful coordinators were defined as people who did not maximize any of their windows, having more than one window visible simultaneously.

Visualisation Tools

Visualisations are powerful tools to gain insights into behaviour and observe certain behavioural patterns (Keim, 2002; Shneiderman, 2001). In particular, large data sets and/or real-time data can benefit from a visual representation (Card et al., 1999). For example, several tools have been developed to visualise how people navigate in and between websites (Chi, 2002; Cugini and Scholtz, 1999; Eick, 2001; Hong and Landay, 2001). Visualisations can, for example, aid in finding certain data, by making relevant groupings, can cause certain data patterns to reveal themselves and allow inferences otherwise unavailable (Card et al., 1999).

Several researchers have specifically investigated visualisation interfaces for temporal and spatiotemporal data. Plaisant *et al.* (1996) describe one of the earliest temporal visualisation systems, which allowed users to browse patient records and extract salient details. Many subsequent projects have also explored temporal visualisation in healthcare (e.g. Bade et al., 2004; Fails et al., 2006; Mamykina et al., 2004; Shahar and Cheng, 1999). In general, the support provided by these systems is strongly tied to their underlying domain, although the 'information visualisation mantra' of 'overview, zoom and filter, details on demand' (Plaisant et al., 1996) is robustly applied to help users move from high level aggregate views of the entire history to specific events in temporal space.

Probably the most similar prior research to our own is by Chi (2002), in which he describes a hierarchical tree view

that captures long term navigational behaviour on the web. Although our objectives are similar (long term navigation across windows rather than the web), the resultant visualisation tools are very different due to the types of data spaces navigated. The web lends itself to hierarchy; windows to 2D space. Plaisant (2004) observes that this phenomenon of slight domain changes demanding substantial changes in design and evaluation is common in visualisation research.

WINDOW WATCHER: DESIGN OBJECTIVES

Visualisation systems are designed to help users gain insights into the underlying properties of the data, yet the design of the system influences the insights that emerge. This is a variant of the Sapir-Whorf hypothesis (Carroll, 1956), and it demands that designers think carefully about the types of observations they hope to reveal, the tasks users might have, the nature of the underlying dataset, and the influence that design decisions will have on users.

In this section we describe the high level design objectives for *Window Watcher*. These objectives are based around four dimensions of the underlying data: space, time, applications and tools. Understanding these dimensions allows us to consider how users may want to query the dimensions or the interactions between them, and to think about how the interface can support them.

Space

Screen real estate is precious. Even with large multi-monitor environments, users can need more display space than there is available. Grudin (2001) conducted an 18 participant field study of window management with multi-monitor environments, but apart from these valuable 'snapshot' findings, we know relatively little about how window desktop space is used. *Window Watcher* must, therefore, support spatial representations of activities with, and between windows.

Space in this context is a three dimensional concept. Windows are explicitly positioned in 2D space by the user by manipulating their origin, size, and aspect ratio. There are interesting insights to be gained in how and when these manipulations occur, whether they occur consistently across applications, and whether spatial zones are used for particular applications and groups of applications. 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. How is depth and window layering managed? What stereotypical strategies emerge?

Window Watcher users will need tools to observe how the 3D screen real estate is used, both in real time and aggregated across different applications.

Time

Window focus policies demand that input be directed to only one window at a time. How do users partition these temporal activities? A previous study suggests that 'window thrashing' occurs (Mackinlay and Royer, 2004), with users executing frenetic short term bursts of window management. Is this a widespread phenomenon, and if so, can the target configuration be characterised (which might suggest that a shortcut could be supported)? Are there temporal patterns of activities between space, applications and their windows, and window switching tools?

Window Watcher users will need tools to observe how windowing activities occur across time. As we are interested in longitudinal analysis, a variety of methods will need to support time aggregation: from time expansion to help scrutinise the millisecond to millisecond state transitions in low level activities such as pointing, clicking, and button presses (e.g. Alt+Tab list traversal), through to extensive time compression to allow a visual overview of months worth of data at a glance.

Applications and their windows

Different applications are used with different frequency and may be used in different ways. For example, prior research (Tak et al., 2009) has demonstrated that the Pareto Principle approximately holds for the frequency of application use, with less than 20% of applications accounting for more than 80% of window switching activities. Furthermore, Grudin's (2001) field study of multi-monitor use showed that some users maintained certain applications like email readers in consistent spatial locations on the screen. How do users use particular applications across time and space?

Another important issue is how applications are used in relation to one another. Several research projects have proposed grouping windows and applications together based on their concurrent use for particular tasks, e.g. a word processor and a bibliography tool (Oliver et al., 2006; Smith et al., 2003), but there is a lack of data on how groupings are used and how frequently applications outside possible grouping are used.

Window Watcher must support users in filtering the data to only display specific applications or groups of applications.

Window switching tools

Finally, users currently have many alternative methods for window switching: clicking within a window, selecting iconic representations such as on the Windows taskbar, cycle through the *z*-order with Alt+Tab (or equivalent), use specialised tools such as Windows Flip 3D or Mac OS X Exposé, or (re)launch the application via the Start Menu, Mac OS X Dock, or a desktop icon. Users may exhibit stereotypical patterns of behaviour signifying the relative utility of these alternatives in different contexts – for example, direct window selections may be less efficient when many windows are visible.

Window Watcher must support users in determining what activities were used to switch windows, and support them in appropriately filtering the dataset.

THE *WINDOW WATCHER* TOOL

In this section, we describe *Window Watcher*'s development and showcase some of its features. Figure 1 shows a mockup of Window Watcher. Its features include

a playback window, a heatmap, a scatter plot, relevant statistics, filtering for the playback window and temporal data and controls.

The development of *Window Watcher* was paired with that of a logging tool called *PyLogger*, which collects low level event data describing window switching activities. In this section, we describe some of *Window Watcher's* specific tools for analysing window switching activities.

The examples used in this section 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 greyscale). *PyLogger* is briefly described in the last section.

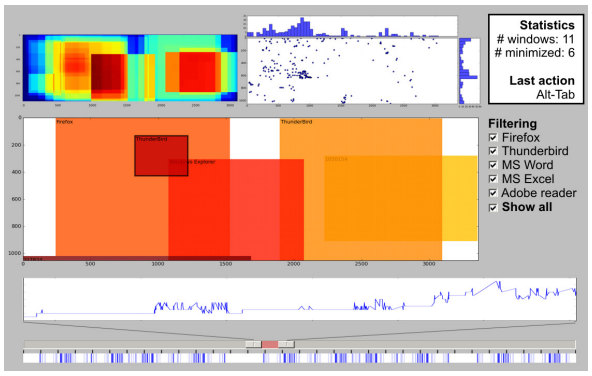


Figure 1. A mockup of Window Watcher. The interactive components shown have been implemented, but are not yet all integrated into the system.

Playback window

Window Watcher's main 'playback' window provides a view of the full extent of the user's display space. The size and shape of the playback window is dynamically reconfigured to portray display changes in the logged data. This occurs when, for example, the logs record the user connecting or disconnecting an external monitor on their laptop.

Figure 2 shows a user who has two screens, both with a resolution of 1680×1050 pixels. The logged user's main screen (with the Windows taskbar) is on the left side, as indicted by the solid red bar along the bottom. All windows (and the Windows taskbar) are shown in their actual locations.

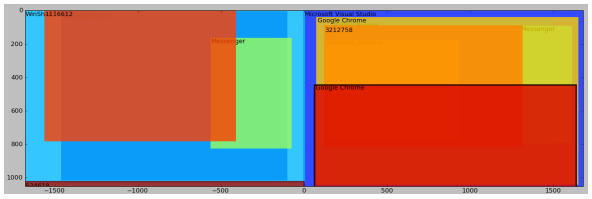


Figure 2. An example of the playback window.

Window *z*-order is essentially the 'depth' of windows on the screen, but more precisely, it is the temporal order in which windows most recently received the window focus. *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 application is shown.

During playback, the content of the logged user's display is continuously updated to reflect changes in their windowing state.

Design issues for temporal data

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.

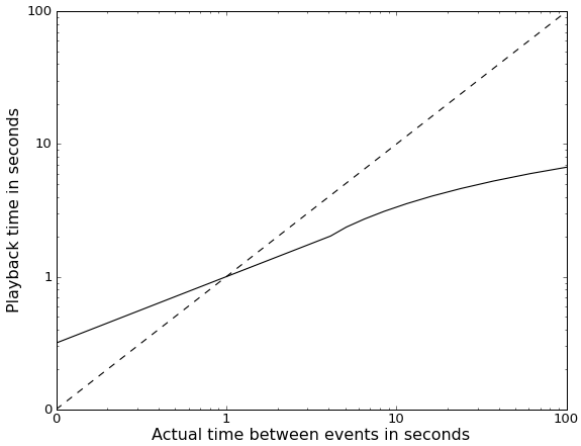


Figure 3. Actual time between events and the playback time in seconds using the conversion described in this paper (dotted line is real time playback). Note both x and y scales are logarithmic.

In addition to manual control of the timeline using standard direct manipulation controls (stop, play, fast forward, etc.), the playback can be event-driven: every window use event (window focus or geometry manipulation) is displayed after a user-configurable delay, to millisecond granularity. This type of playback helps a viewer gain an overall impression of window management activities, but the details of bursts of window switching activities are still hard to perceive.

However, with this type of playback detailed information about the nature of window interaction can be lost, such as the burst-like characteristic of window use activities. We therefore also include a time conversion mechanism, 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. Our algorithm converts the time between two events as follows: if the time *t*

between 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 3. The conversion leads to the desired effect: slowing down small values and speeding up (very) large values.

We are currently implementing two further interactive controls: an option based on the algorithm above that automatically calculates the parameter values in order to complete with entire selected playback in a specified period of time (e.g. the viewer selects 'play back in 2 minutes'); and a jog-shuttle gesture based input using a Wacom Tablet.

Temporal data

A timeline view, shown in Figure 1 at the bottom, provides an overview of daily window switching events. Each vertical line represents a window use event (either the top window has changed or the top window has been moved/resized). The scaling of the x-axis is linear. We are currently implementing a two-level semantic-zooming timeline view, motivated by the Google Finance timeline browser (www.google.com/finance). Users will be able to control the range of the playback using a bidirectional slider as well as seeing a semantically appropriate summarisation of the data dependent on the view granularity.

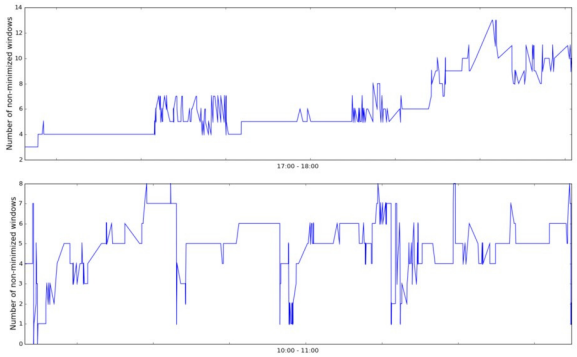
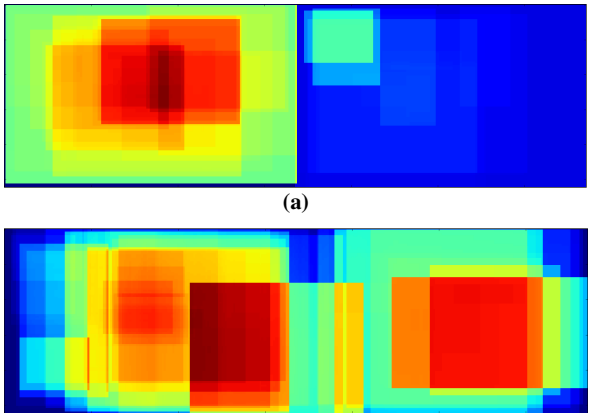


Figure 4. Two plots of the number of non-minimized windows over one hour.

An example of such a summarisation is shown in Figure 4. Here, the number of non-minimized windows over one hour is plotted for two different participants. The top plot shows gradual increases and decreases in the number of non-minimized windows, while the bottom plot shows sudden increases and decreases in the number of non-minimized 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-minimized 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.

Heatmaps

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, and by certain applications. Also, it can help to identify certain use patterns. For example, Figure 5 shows heatmaps of two days of data for two different users, both of whom use dual 1680×1050 monitors. The user shown in Figure 5a has a clear preference for the left screen, while the user in Figure 5b uses both more or less equally. Also, the user in Figure 5a often has a window maximized in the left screen, while the user in Figure 5b does not seem to maximize windows often.



(a)

(b)

Figure 5. Heatmaps for two different users.

Visualising window switching techniques

Window Watcher also portrays the method used to switch windows by displaying the location of mouse clicks prior to window switching activities. Figure 6, for example, shows a snapshot of the playback window when the user switches to Thunderbird using the Windows taskbar.

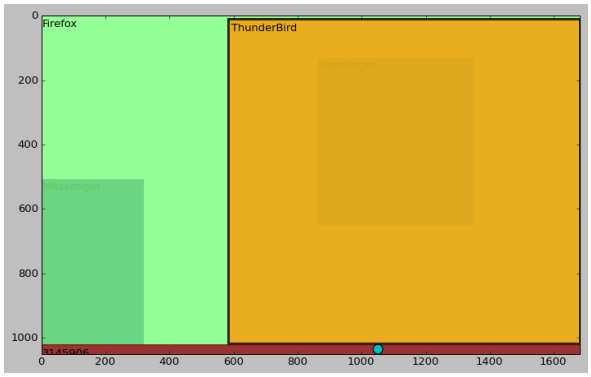


Figure 6: The location of mouse clicks is portrayed with a cyan circle, suggesting the method used to switch windows (in this case, the Windows taskbar).

Summary statistics and frequency data for all window switching methods (including non mouse-based window switching tools such as Alt+Tab) are shown in summary statistics elsewhere in *Window Watcher*. Figure 7 shows a summary statistics plot, revealing the location of a range

of mouse clicks made by one user. Clicks on the taskbar have been filtered out by the user (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 6), he/she often clicks on the title bar of the window to switch to this window (see Figure 6 and 7), even though a click *anywhere* in the window would bring the window to the focus.

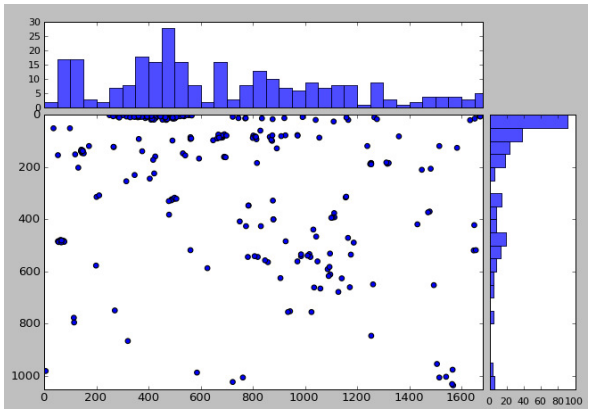


Figure 7. A scatter plot and two histograms visualising the locations of mouse clicks (Windows taskbar clicks omitted).

Examples of insights gained with *Window Watcher*

We have given several examples of how *Window Watcher* informs us about window switching activities in a manner that probably would not be evident from raw statistics alone. However, as Perer and Shneiderman (2008) observe quantitative statistics and visualisations are often most powerfully manipulated in combination, and we have observed this effect in our own analysis.

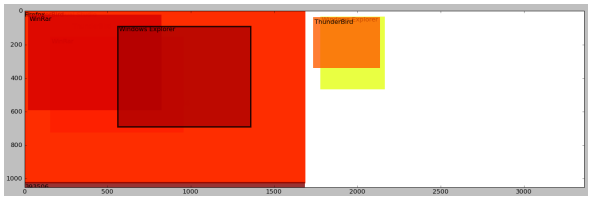


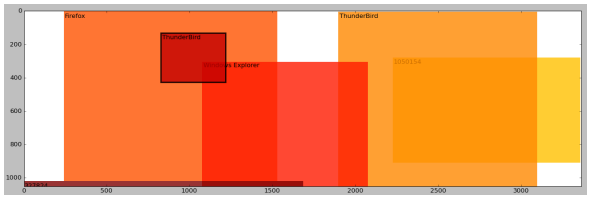
Figure 8. Playback window for a user with an unusually high number of windows open in the left screen, but a maximised window obscuring them.

For example, our logs showed one participant who had an unusually high mean number of windows open (23.7). When browsing this participant's data using *Window Watcher* it quickly becomes apparent that the user was a 'stacker', using one of their two screens almost to the exclusion of the other, and maintaining a 'neat' (but deep) pile of windows, normally with the top one maximised. Figure 8 shows a representative *Window Watcher* snapshot for this user. Even though this user has 29 windows open (and 11 non-minimized) at this particular moment, the screen does not look cluttered.

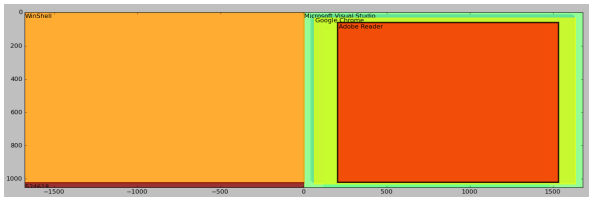
Window Watcher has also been useful for discriminating between radically different usage styles, even when summary statistics suggest they might be similar. For example, Table 1 shows statistics for two different users who have similar averages for the total number of windows, number of non-minimized windows and number of windows (partly) unobscured (accumulated over a three week period). However, when the data of these participants is viewed with *Window Watcher*, two very different windowing management styles emerge. Figure 9 shows two representative screenshots for P1 and P2 (Figure 9a and b respectively). P1 organises windows in a way that windows are hardly ever maximized and many windows have a relatively large part visible. P2 stacks windows on top of each other, but adapts the size of the window so that a (small) portion of the underlying window is still visible. Neither of these windowing behaviours are evident from the raw data.

	P1	P2
Average # windows	15.1	14.2
Average # non-minimized windows	8.6	10.8
Average # windows (partly) visible	6.9	6.3

Table 1. Two similar data sets



(a)



(b)

Figure 9. Visualizations of two participants with similar summary statistics.

LOGGING WINDOW SWITCHING WITH *PYLOGGER*

We developed a logging tool called *PyLogger* to collect window switching data in an unobtrusive manner. *PyLogger* records when a new window gets focus and what action caused it (e.g., a click on a taskbar button, or a new program launched using a Quick Launch button). It also records when the focal window is moved or resized. Lastly, it keeps track of the position and states (e.g., minimized) of all windows. *PyLogger* runs silently in the background and has no noticeable impact on the computer's performance.

PyLogger uses a polling loop to check every 100 milliseconds whether the focal window has changed. A change in the focal window is defined as either a focus redirection or a change in the size/position of the focal window. Meanwhile, a *lastAction* variable keeps track of the last user action, such as a left mouse button click

(including the position of the cursor at the time the button was clicked) or a keystroke. Some examples of the values the *lastAction* variable can have are **keyboard ['Alt', 'F4']** (the key combination Alt+F4), **left [216, 768]** (the left mouse button was clicked at screen coordinates (216, 768)) or **ltaskbar [6,9]** (the 6th button on the Windows taskbar was clicked with the left mouse button, and there were 9 buttons visible on the taskbar). Whenever the polling loop detects a change the output method is called, which writes the following information to the output 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;
- How many windows are open (including minimized windows);
- Information about the focal window: window handle, window title, window position 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.

Figure 10 shows an example of a section of a *PyLogger* log file, which is easy for automated scripts to parse.

```
2009-02-05 17:22:34.900000 || NEWTOPWINDOW || ltaskbar || [6,
9] || 11 || (1901970, 'National: press.co.nz - Mozilla
Firefox', (1719, 47, 3056, 968), 'MozillaUIWindowClass') ||
[(393312, '', (0, 1020, 1680, 1050), 0, 'Shell_TrayWnd'),
(1901970, 'National: press.co.nz - Mozilla Firefox', (1719, 47,
3056, 968), 0, 'MozillaUIWindowClass'), (1245446, 'PyLogger',
(3061, 11, 3269, 168), 0, 'wxWindowClassNR'), (524636,
'Document1 - Microsoft Word', (0, 0, 1561, 970), 0, 'OpusApp'),
(1114486, 'output.txt - Notepad', (231, 190, 1616, 1020), 0,
'Notepad'), (2752766, 'logger', (428, 132, 1228, 732), 0,
'ExploreWClass'), (262416, 'polledlogger.py -
H:\\python\\logger\\polledlogger.py', (129, 43, 808, 760), 0,
'TkTopLevel'), (591518, '*Python Shell*', (232, 155, 1492,
872), 0, 'TkTopLevel'), (262856, 'Inbox - Thunderbird', (51, 4,
1638, 995), 0, 'MozillaUIWindowClass'), (1574258,
'logger_analysis.py - H:\\python\\logger\\logger_analysis.py',
(-32000, -32000, -31840, -31969), 0, 'TkTopLevel'), (65678,
'Program Manager', (0, 0, 3280, 1050), 0, 'Progman')]
```

Figure 10. An example of PyLogger output

In total, 26 people installed *PyLogger* on their computer, for a period of approximately three weeks. The number of hours per week the participants used their computer ranged from 8 to 90 hours, with an average of 47 hours. Seven participants used Windows Vista, the rest used Windows XP. Ten participants used a single monitor; nine used dual-monitor setups; and seven used a mix of both (e.g., a laptop that is sometimes extended with an extra monitor).

Data filtering

Not all items that are *technically* windows are actually windows that the user can interact with like any other (regular) window. For example, the Windows taskbar and the Windows desktop are in fact windows (see Figure 10, the taskbar and desktop have window classes 'Shell_TrayWnd' and 'Progman', respectively). For numerical analysis, these items should not be counted as windows, as this would return an incorrectly high number of windows. With a visualisation tool, these 'special cases' in the data are easily identified.

DISCUSSION, FURTHER WORK, CONCLUSIONS

Development work with *Window Watcher* is ongoing as we use it to help us understand the logs generated in our longitudinal analysis of window switching activities. However, we have already been surprised at how much additional information we have gained from its use. Statistical summaries are useful (and are accessible via *Window Watcher*), but it is the *combination* of quantitative statistics with visual replays, heatmaps, and semantic filtering that has proven most useful. This observation was recently made by Perer and Shneiderman (2008) in their study of a political analyst, bibliometrician, healthcare consultant, and counter-terrorism researcher, but it is the first time it has been made (to our knowledge) in the core HCI business of understanding interaction with computer systems.

The general 'silk from a sows ear' problem (Pirolli et al., 1996) of extracting end user characterisations from activity logs is an important one for understanding how systems can be iteratively improved. It is likely to become more important as windowing toolkits become better able to support automated logging (Alexander and Cockburn, 2008) and as more companies deploy customer experience improvement programs (e.g. www.microsoft.com/products/ceip).

Window Watcher was specifically designed to aid visualisation of spatiotemporal data, which is the essence of window management, as well as supporting a variety of features for filtering based on time and salient window management activities (the applications used and the window switching mechanisms). However we believe some of its 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).

Further work will focus on using *Window Watcher* to generate an empirical characterisation of window switching behaviour, and to use this characterisation to generate new and improved window switching interfaces. Once complete, we will release *Window Watcher* and *PyLogger* as open source code.

ACKNOWLEDGMENTS

This work was partially funded by New Zealand Royal Society Marsden Grant 07-UOC-013.

REFERENCES

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

Bade, R., Schlechtweg, S., Miksch, S. Connecting time-oriented data and information to a coherent interactive visualization. Proc. CHI '04, ACM (2004) 105-112.

Bernstein, M.S., Shrager, J., Winograd, T. Taskposé: exploring fluid boundaries in an associative window visualization. Proc. 21st annual ACM symposium on User interface software and technology, ACM (2008) 231-234.

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

Bly, S.A., Rosenberg, J.K. A comparison of tiled and overlapping windows. Proc. CHI '86, ACM (1986) 101-106.

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

Carroll, J.B. (ed.): Language Thought and Reality: Selected Writings of Benjamin Lee Whorf. The MIT Press (1956)

Chi, E.H. Improving Web usability through visualization. Internet Computing, IEEE 6, 2 (2002) 64-71.

Cugini, J., Scholtz, J. VISVIP: 3D visualization of paths through web sites. Proc. 10th International Workshop on Database and Expert Systems Applications, IEEE (1999) 259-263.

Eick, S., G. Visualizing online activity. Commun. ACM 44, 8 (2001) 45-50.

Fails, J.A., Karlson, A., Shahamat, L., Shneiderman, B. A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events across Multiple Histories. IEEE Symposium On Visual Analytics Science And Technology, IEEE (2006) 167-174.

Gaylin, K.B. How are Windows Used? Some Notes on Creating an Empirically-Based Windowing Benchmark Task. Proc. CHI '86, ACM (1986) 96-100.

Grudin, J. Partitioning Digital Worlds: Focal and Peripheral Awareness in Multiple Monitor Use. Proc. CHI '01, ACM (2001) 458-465.

Hong, J.I., Landay, J.A. WebQuilt: a framework for capturing and visualizing the web experience. Proc. 10th international conference on World Wide Web, ACM (2001) 717-724.

Hutchings, D., Smith, G., Meyers, B., Czerwinski, M., Robertson, G. Display Space Usage and Window Management Operation Comparisons between Single Monitor and Multiple Monitor Users. Proc. AVI'04, ACM (2004) 32-39.

Hutchings, D.R., Stasko, J. Revisiting display space management: understanding current practice to inform next-generation design. Proc. GI 2004, Canadian Human-Computer Communications Society (2004) 127-134.

Keim, D.A. Information visualization and visual data mining. IEEE Transactions on Visualization and Computer Graphics 8, 1 (2002) 1-8.

Kellar, M., Hawkey, K., Inkpen, K.M., Watters, C. Challenges of Capturing Natural Web-based User Behaviours. International Journal of Human-Computer Interaction 24, 4 (2006) 385-409.

Kumar, M., Paepcke, A., Winograd, T. EyeExpose: Switching Applications with Your Eyes. Stanford University (2007)

Mackinlay, J.D., Royer, C. Log-based Longitudinal Study Finds Window Thrashing. Palo Alto Research Center (2004)

Mamykina, L., Goose, S., Hedqvist, D., Beard, D.V. CareView: analyzing nursing narratives for temporal trends. Proc. CHI '04 extended abstracts, ACM (2004) 1147-1150.

Myers, B.A. A taxonomy of window manager user interfaces. IEEE Computer Graphics and Applications 8, 5 (1988) 65-84.

Oliver, N., Smith, G., Thakkar, C., Surendran, A. SWISH: Semantic Analysis of Window Titles and Switching History. Proc. 11th international conference on Intelligent user interfaces, ACM (2006) 194-201.

Perer, A., Shneiderman, B. Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. Proc. CHI '08, ACM (2008) 265-274.

Pirolli, P., Pitkow, J., Rao, R. Silk from a Sow's Ear: Extracting Usable Structures from the Web. Proc. CHI'96, ACM (1996) 118-125.

Plaisant, C. The challenge of information visualization evaluation. Proc. AVI '04, ACM (2004) 109-116.

Plaisant, C., Milash, B., Rose, A., Widoff, S., Shneiderman, B. LifeLines: Visualizing Personal Histories. Proc. CHI '96, ACM (1996) 221-227.

Robertson, G., Horvitz, E., Czerwinski, M., Baudisch, P., Hutchings, D., Meyers, B., Robbins, D., Smith, G. Scalable Fabric: Flexible Task Management. Proc. AVI '04, ACM (2004) 85-89.

Shahar, Y., Cheng, C. Intelligent visualization and exploration of time-oriented clinical data. Topics in health information management 20, 2 (1999) 15-31.

Shneiderman, B. Inventing Discovery Tools: Combining Information Visualization with Data Mining. Discovery Science (2001) 17-28

Smith, G., Baudisch, P., Robertson, G., Czerwinski, M., Meyers, B., Robbins, D., Horvitz, E., Andrews, D. GroupBar: The TaskBar Evolved. Proc. OzCHI '03, (2003) 34-43.

Tak, S., Cockburn, A., Humm, K., Ahlström, D., Gutwin, C., Scarr, J. Improving Window Switching Interfaces. Proc. Interact '09, Springer-Verlag (2009) 187-200.