Command Selection

Gilles Bailly, Sylvain Malacria

Command Selection

Gilles Bailly and Sylvain Malacria

Contents

Abstract

Command selection is probably one of the most ubiquitous and important tasks with interactive systems. It consists in offering to the users one or several methods to select a command among a set of commands. These methods include menus, toolbars, keyboard shortcuts, etc. In this chapter, we first categorize and describe the main methods for selecting commands. We then discuss how certain platforms introduce opportunities and constraints regarding command selection. After that, we detail key features that designers should consider when designing methods for selecting commands and how these features can be adapted during the interaction. We then review how to improve performance when selecting commands. Finally, we highlight three main challenges for future work on command selection.

Keywords

Menus · Shortcuts · Commands · Organization · Presentation · Navigation · Selection

Introduction

We frequently open applications on our smartphones, navigate within the hierarchy of a website, select commands through menus in word processors and spreadsheet applications, or perform keyboard shortcuts. All these tasks are related to command selection.

Command selection is one of the most fundamental tasks in human-computer interaction (HCI). It concerns all interactive systems from smartwatches to public displays through desktop workstations and smartphones. For instance, menus are one typical method for selecting commands on all these systems, but several others are available (e.g., toolbars, keyboard shortcuts, etc.). Depending on the context or system, they rely on different input devices (touch, mouse, keyboard, speech input) or output modalities (e.g., visual list of text labels, grid of icons, speech, etc.) but serve the same purpose: presenting available commands to the users and letting them discover, navigate, and/or execute these commands.

The task of selecting commands is so important that it frequently stimulates innovative designs in commercial products in an attempt to improve usability and performance of user interfaces (e.g., Xerox Star in 1981, the Apple iPod wheel menu in 2001, the radial menu in the video game Secret of Mana from Square in 2003, the Microsoft Ribbon, or the iPhone application Menus in 2007). Saving even a few milliseconds off a frequent operation can result in significant cumulative savings when considering a complex task (e.g., 3D modeling) with implications on productivity. Similarly, subtle well-designed features in a menu such as graphical icons, audio cues, or animations can improve usability, create curiosity, and provoke enjoyable experience which are essential in games and entertainment applications (Lazzaro 2009).

Command selection is also an important proxy to study HCI. While it is only one facet of HCI, it covers many fascinating and challenging phenomena. One single user interface can lead to so many different users' behaviors that it is often too complex to precisely identify and study each factor. In contrast, a simple pointing task (which is already quite complex) often overlooks fundamental aspects such as those related to visual search, skill acquisition, decision-making, cognitive bias, etc. The nature of command selection as an interaction mechanism that is neither too analytic nor sensory-based makes it an exciting area of research with uninterrupted progress for more than 60 years in terms of interaction design, empirical findings, and computational models.

In this chapter, we address the following questions:

- What is command selection and what are the main methods for selecting commands?
- What are the design factors of command selection?
- How to adapt command selection to the user?
- How to improve users' performance when selecting commands?

Command Selection

What Is Command Selection?

While there is no general agreement on the exact meaning and scope of command selection, we propose to define command selection as the task that consists in choosing one specific *command* among a set of commands. A *command* is defined as a change in the internal state of the application, typically opening a file, launching an algorithm, starting a network connection, etc. Users then need interaction techniques to trigger a command. An *interaction technique* is a "*combination of input and output, consisting of all software and hardware elements, that provides a way for the user to accomplish a task*" (Tucker 2004).

Following these definitions, we call an *interaction method* a class of interaction techniques sharing key features (such as using permanent screen space) that we discuss now.

Methods for Selecting Commands

The second question that we must anticipate is "What are the main methods for selecting commands?". Several methods such as menus, toolbars, keyboard shortcut, and command line interfaces are often available in a given application, and each of them has different advantages and drawbacks. However, precisely answering this question is challenging due to the lack of clear terminology and categorization of existing methods for command selection. For instance, while everyone has a raw idea of what a "menu" is, there is no consensual definition of this term in

the literature (Bailly et al. 2016), and the wordings and/or the scope might differ between research articles, commercial products, and programming tool kits.

It is common in the literature to classify existing methods by dichotomy, for instance, novice vs. expert methods (Cockburn et al. 2014; Bailly et al. 2016), menus vs. shortcuts (Kurtenbach and Buxton 1991; Grossman et al. 2007), navigation-based vs. direct methods (Shneiderman et al. 2016), and recognition-based vs. recall-based methods (Lee and Raymond 1993). These dichotomies simplify the problem and highlight key aspects of command selection such as the envisioned targeted users (novice vs. expert), the performance (menu vs. shortcuts), or cognitive considerations (recognition-based vs. recall-based). However, they also tend to mask subtle but important aspects of command selection. For instance, there is no strong relation between users' expertise (novice/expert) and the choice of the method (Lafreniere et al. 2017; Kurtenbach and Buxton 1994; Bailly et al. 2016): in spite of significant experience, many users continue to use menus, toolbar, etc. and do not use keyboard shortcuts (Lane et al. 2005). Many users also switch back and forth between menus and shortcuts, for instance, after a long lay-off period (Kurtenbach and Buxton 1994). Moreover, the recognition-based vs. recall-based dichotomy suggests that some methods are "recall-based" by nature, while they are often "recall-based" by design. For instance, Malacria et al. demonstrated that keyboard shortcuts, almost always defined as a recall-based method, can be efficiently used in a recognition-based manner that does not require to memorize the shortcut beforehand (Malacria et al. 2013).

In this chapter, we do not contrast methods for selecting commands using one of these dichotomies. Instead, we distinguish and describe five popular methods that differ in key properties.

Command Lines Before the introduction of graphical user interfaces (GUIs), command line interfaces (CLIs; Fig. 1a), also called command languages, were the main method for executing commands. The user types a command name and its different parameters and then presses the Enter key to execute the command and observe the result. This method requires users to be aware of the available commands and their associated parameters beforehand. While this is obviously a barrier to interacting with this method, CLIs remain powerful for selecting commands once users know them (Murillo and Sánchez 2014; Sampath et al. 2021; Barrett et al. 2004; Hendy et al. 2010). It is probably why they are often adopted by power users such as system administrators (Murillo and Sánchez 2014).

Menus A menu (also called pop-up menu) is probably the most common widget for selecting commands (Bailly et al. 2016) in GUIs (Fig. 1b). It generally has a linear layout and contains an internal structure to organize the different items (see section "Organizing Items"). Menu items typically contain a text label (command name), an icon, and/or a keyboard shortcut cue. Sometimes, they can include additional symbols or widgets such as toggle buttons or checkbox (see section "Item"). Moving the cursor over a menu highlights the item below the cursor, while clicking

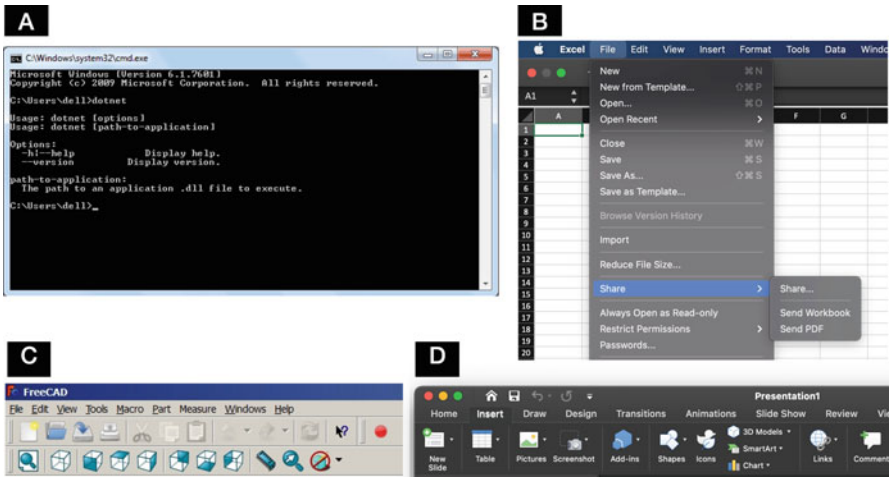

Fig. 1 Examples of methods for selecting commands. (**a**) Command line interfaces (CLIs); (**b**) pull-down menus attached to the menubar; (**c**) toolbar; (**d**) Ribbon

executes it (and disposes the menu). A key characteristic of a menu is to be transient: they appear on demand and are closed immediately after the selection of an item and thus do not require permanent screen space. It exists different classes of menus in GUIs:

Hierarchical menus (also called cascaded menus or pull-down menus). Some menu items can, instead of activating commands, be used to access to so-called submenus: the user clicks or hovers an item with a pointing device (or finger), and the corresponding submenu is displayed.

Context menus are a specific class of menus that contain different items depending on the context. Typically, a user right-clicks on an object of interest which displays, at the click location, the context menu. This one presents only a subset of the commands that can be applied to this object. By filtering out nonrelevant options, context menus reduce visual search and use less screen space.

Menu bars are generally permanently displayed, typically at the top of each application window (on Windows or Unix-like operating systems), the top of the screen (on macOS), or the top or side of a web page. They contain a limited set of items (about nine (Malacria et al. 2013)), which generally only have a category name (no icon). Typical category names are File, Edit, View, Window, and Help. These items give access to a hierarchy of menus.

Mnemonics are a keyboard-based method that relies on a sequence of key presses to navigate menus (or Ribbons; see section "Methods for Selecting Commands") and eventually activates a desired command. Typically on Microsoft Windows, mnemonic mode is activated by typing the Alt modifier key, which underlines exactly one character of each visible item of the menu (Fig. 3). Users

select an item of the menu, for instance, a submenu, by pressing the corresponding mnemonic key which in return opens that submenu and displays all of its elements with exactly one character underlined, and so on. Mnemonics were initially introduced as an accessibility feature of the interface as reflected by their name in MS Windows, access keys.

Toolbars are graphical widgets *permanently* displayed at the top of an application window or right below the menubar and providing a direct access to commands (Fig. 1c). Unlike menus, they consume screen space that cannot be used for the rest of the application. In return, they facilitate both the visual inspection and the selection of available items as they do not require a first action to display the widget. Toolbar items are usually organized in a grid layout. To reduce the total amount of screen space occupied, it is common to use items with small icons and/or with text labels and to present only a subset of the commands of the application.

Palettes are a flexible type of toolbars that users can move and sometimes resize. In particular, the users can move them so that it reduces the distance, and thus the time taken, for accessing the items from the current object of interest (Hascoët et al. 2006).

Ribbons are another type of toolbar that have been introduced by Microsoft in its Office Suite in 2007 (Fig. 1d). They can be described as "tabbed" toolbars. Basically, they organize several toolbars semantically, each in its specific tab, and clicking on a tab displays the corresponding toolbar. Ribbons thus extend toolbars by providing a top-level hierarchical organization.

Page-based menus Similar to Ribbons, page-based menus (or CardLayout menus), which are not really menus as they use a permanent space, are a variant where two or more menus share the same display space. A typical example is the iOS home screen (or Launchpad) that displays a grid of items that users can tap on to launch the corresponding applications. Users can also perform a horizontal swipe to navigate through the different pages.

Shortcuts Shortcuts provide a direct access to commands. We distinguish keyboard and gesture shortcuts.

Keyboard shortcuts allow users to execute a command without having to travel a menu hierarchy (in contrast to mnemonics that require to navigate the menu hierarchy). A typical keyboard shortcut requires the user to hold one or more modifier key(s) (e.g., Ctrl, Shift, Alt, cmd on macOS) and press a hotkey (an alphanumeric key such as "P," "7," or "/"). These modifier keys are used to discriminate keyboard shortcuts from general keyboard input such as text entry. However, keyboard shortcuts can also be used without modifiers (only hotkey press) in certain applications where the keyboard is not by default used for text input. Examples can be found in graphics editing software (e.g., Adobe Illustrator or GIMP) to switch between tools (Note that the word hotkey is also commonly used to refer to keyboard shortcut.).

Keyboard shortcuts are fast to execute and can be performed with the left hand while the right hand operates the pointing device. In applications where users

frequently type text, keyboard shortcuts allow the execution of commands without having to move the hand back and forth between the keyboard and the mouse. However, keyboard shortcuts generally require some efforts to discover and learn the mapping between the commands and the corresponding key combinations. Current applications generally (but not always) reveal this mapping in menus (a keyboard shortcut is displayed on the right side of the items), via tooltips when the mouse cursor hovers a toolbar button, or in a dedicated window (cheat sheet) listing all keyboard shortcuts and their corresponding commands.

Interestingly, modern tablet computers also provide keyboard shortcuts (Fennedy et al. 2020). For instance, on iPads, they are executed on a physical (Bluetooth) keyboard and communicated via a pop-up window appearing when the cmd key is held for a few seconds. On Microsoft tablets, keyboard shortcuts are communicated and executed directly on the virtual keyboard: Hitting or holding a modifier key (Ctrl) displays the command name on the corresponding hotkey that the user can hit to execute the command.

Gesture shortcuts With the increasing number of interactive surfaces (touch pad, touchscreen, etc.), gesture shortcuts are more and more available. For instance, multi-touch touch pads provide various gesture shortcuts to zoom, scroll, list applications, reveal windows, etc. They share several similarities with keyboard shortcuts: they provide a direct access to the commands, are fast to execute, and might be difficult to discover and learn. However, Appert et Zhai found that, while they have the same level of performance, gesture shortcuts are easier to learn and recall (Appert and Zhai 2009). Gesture shortcuts also have differences. They do not require a keyboard; they are likely to provide a more fluid navigation experience but also to be triggered inadvertently. Moreover, gesture shortcuts typically provide access to system-wide commands and, as such, do not need to be associated with application specific commands located in menus or toolbars. As a result, they are not signified in an application GUI, for instance, near a menu item, and must be learned from a dedicated interface in the OS typically the touch pad configuration panel.

The Diversity of Methods

In the previous subsection, we provide an overview of the most common methods for selecting commands. Interestingly, there is often more than one available method for selecting commands in a given application, and the user can freely choose the preferred one(s). These methods can also be combined, extending the design space of available interaction techniques. For instance, toolbars can be augmented with menus: some toolbar items are hierarchical and display a menu when clicked. Reciprocally, a Ribbon can be seen as a menubar opening several toolbars. Some methods can also be transformed. For instance, several toolbars can be detached and moved such as palette. Similarly, a Tear-off menu is a menu that a user can detach from the menubar to transform it into a palette.

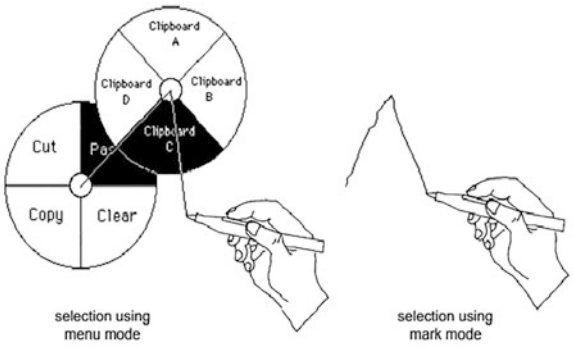

Fig. 2 Hierarchical Marking menus

There are several situations where the frontier between methods is even more fuzzy. For instance, Marking menu (Kurtenbach 1993) is a famous gesture-based interaction technique available in some Autodesk software (Fig. 2). When the user presses down the pointing device and waits for a certain delay (about 333 ms), the menu appears centered around the position of the cursor, allowing item selection by moving in the direction of the desired item; if the user does not wait and begins dragging immediately, the menu is not displayed, and the cursor draws a mark. When the user releases the mouse, the gesture recognizer determines the selected item. Therefore, Marking menus somewhat combine a radial menu with gesture shortcuts. Another example is *crossing-based* methods (Accot and Zhai 2002; Apitz and Guimbretière 2004; Fruchard et al. 2020) allowing users to select a command by crossing (sometimes entering) an item with the cursor instead of clicking on it. A variant consists of selecting a command and its parameter in the same action making it difficult to distinguish command selection and direct manipulation (Guimbretiére and Winograd 2000; Pook et al. 2000; Apitz and Guimbretière 2004).

In conclusion, this section illustrates the diversity of existing methods for selecting commands and how they are inter-connected. We defined the most common ones and their key, often subtle, properties. It remains that precisely delimiting the scope of each of them is challenging and probably unnecessary.

Platform, Application, Input, and Output Modalities

The previous methods are not available on all platforms, and their implementation varies depending on the system, the device, and/or the application. In this section, we discuss different factors from the environment that influence the implementation and the interaction with these methods.

Operating System

Methods for selecting commands (menus, shortcut, etc.) share many similarities between operating systems. However, each operating system has its own design guidelines that can have an impact on the interaction and the user experience.

A notorious difference is the location of the menubar under Microsoft Windows and Apple macOS. If three windows are opened on Windows, three menubars (one per window) are actually displayed. In contrast, macOS has only one menubar visible (at the top of the screen) at a given time, but the content (menu titles) changes depending on the application that has the focus. At first glance, selecting a command in the macOS menubar could appear slower than in the Windows one because it is on average further from the mouse pointer, especially on large screens. The reality is more subtle as the macOS menubar items can be seen as "infinitely large" because the edge of the screen is an "impenetrable border" for the mouse. According to Fitts' law (Fitts 1954), the distance and size of an item influence pointing time. In practice, the temporal difference appears to be small and depends on the initial location of the mouse cursor. Another aspect to consider is screen space. The macOS strategy uses less screen space than the Windows one as there is a single menubar regardless the number of opened applications. However, this strategy can also introduce confusion as users can interact with the current menubar without immediately noticing that it does not correspond to the desired application (because it does not have the focus). This example highlights different strategies to implement the menubar, the variety of criteria to consider (pointing time, screen space, confusion), and the difficulties to choose which strategy is the best one as it depends on the usage (e.g., number of opened applications, screen size).

Keyboard shortcuts also differ between operating systems. For instance, the main modifiers under macOS are the proprietary *command* keys (⌘, e.g., ⌘+C for copy), positioned on each sides of the space bar and inviting to be pressed with a thumb. On the other hand, keyboard shortcuts on Microsoft Windows and Unix-like operating systems rely on Ctrl keys (Ctrl+C for copy) located in the bottom corners of the keyboard layout, thus further from the space bar and inviting to be pressed with a pinky finger. The mapping between commands and keyboard shortcuts also differs between operating systems due to different guidelines. As an example, macOS (https://developer.apple.com/design/human-interface-guidelines/macos/user-interaction/keyboard/) encourages not to use the Ctrl key. Moreover, macOS recommends to use the Alt key as a second modifier sparsely (e.g., ⌘+Alt+S for Save As), while Microsoft (https://docs.microsoft.com/en-us/previous-versions/windows/desktop/dnacc/guidelines-for-keyboard-user-interface-design) recommends to avoid it (F12 for Save As).

Finally, differences also exist regarding the availability of mnemonics for selecting commands that have been long implemented in Microsoft Windows and Unix-like operating systems but are to this date not available on macOS.

Application

Given an operating system, the item location of frequent commands (e.g., "Open" or "Copy") in pull-down menus or their keyboard shortcuts are generally consistent from one application to another. This favors skill transfer when discovering a new application. However, we observe more variability regarding their icons probably

due to some constraints related to the global aestheticism of the application and/or the brand identity (see section "Item").

Interestingly, some applications have their own methods to improve command selection. For instance, Microsoft applications have their proprietary method, Ribbons, described above. Another example is Marking menus (Kurtenbach and Buxton 1991, 1994; Kurtenbach 1993; Henderson et al. 2020) present in some Autodesk softwares (see section "The Diversity of Methods" and Fig. 2). Interestingly, Autodesk software also offers sometimes the Hotbox (Kurtenbach et al. 1999), multiple menubars displayed on demand in the center of the application. Finally, Emacs is a singular application with his own shortcut mechanism: some commands have keyboard shortcuts relying on several sequential key presses rather than a single key chord (e.g., Ctrl+X Ctrl+W for *Save Buffer As...*) due to historical design choices before the introduction of modern GUIs.

Application vs. Web

Hierarchical menus are used both as a method to select commands in software applications or to navigate on websites. However, there are subtle differences between application menus and web menus. For instance, while application menus (or menubars) are generally visible at the top of the window, it is frequent that web menus are not visible when users are scrolling down the web page. The reason is that web menus are an intrinsic component of the code of the web page and only the content within the viewport is displayed. One solution consists in displaying at the bottom of the web page a "go to top" button that scrolls back to the top of the website when activated. Another is to use "floating" web menus that remain always visible at the top of the viewport, regardless of scrolling.

Another difference is that websites often provide much less keyboard shortcuts than desktop applications because the web browser already processes most of them. Moreover, websites are more and more accessed from touch-based devices such as smartphones that do not provide keyboard shortcuts.

Input and Output Modalities

Command selection is impacted by the type of inputs (mouse, keyboard, touch, mid-air gesture, etc.) and output modalities (e.g., the size of the display). For instance, items might be difficult to precisely select on a smartwatch or difficult to reach on very large touchscreens. In contrast, the new input and output capabilities of multi-touch, IMUs, body input, haptic and audio feedback, etc. offer new modalities to exploit for command selection (Fruchard et al. 2018; Bailly et al. 2013b; Dubois et al. 2018; Keddisseh et al. 2021; Buschek et al. 2018; Zheng and Vogel 2016; Walter et al. 2013). In this section, we focus on two use cases: small devices and multi-touch devices. We refer to Bailly et al. extensive literature review (Bailly et al.

2016) for more examples of methods relying on advance input/output modalities for selecting commands.

Small devices The introduction of smartphones and more recently smartwatches emphasized the challenges of designing menus for touch-based input on small screens. Reducing the size of the items to fit the screen size is not an efficient solution as it impairs readability and introduces imprecision when selecting items with a finger. It generally requires a radical rethinking of command selection both in terms of functionalities, presentation, and interaction. For instance, a smartphone application usually does not include all the commands of the desktop one (Wagner et al. 2014) but only the most likely to be used in a mobile scenario and in a given context. While hierarchical organizations are common on desktop applications, a simple scrolling list is sometimes more appropriate to avoid multiple taps which are error-prone.

Permanently displaying a menubar is also complicated because of the limited screen space of these small devices. One strategy used to overcome this issue is the hamburger button (≡), usually located in one of those found at the top corner of the UI and whose function is to toggle a menu (usually called the hamburger menu (Casadei et al. 2017)). Originally introduced in the first desktop operating systems in the 1980s but rapidly removed, it made a thunderous comeback recently as a work-around for the limited screen area in mobile apps.

Finally, the use of direct touch as the main interaction paradigm for these devices also required an alternative to post context menus, typically triggered with a right-click on desktop platforms but triggered either with a long or force press on smartphones and tablets. Another alternative can be found in Swhidgets (Pong and Malacria 2019), commands by default hidden under an interface element of interest and that users uncover with a simple horizontal swipe gesture. While present in numerous applications, it has been shown that both the hamburger button and Swhidgets might suffer from discoverability issues (Pong and Malacria 2019).

Multi-touch input Multi-touch devices (touchscreens, touch pads, etc.) offer interesting interaction opportunities in particular command selection (Lepinski et al. 2010; Bailly et al. 2008, 2012a; Ghomi et al. 2013; Gutwin et al. 2014; Goguey et al. 2019). For instance, FastTap (Gutwin et al. 2014) is an interaction technique that displays items in a spatially stable grid layout when a finger is held on a dedicated menu button. Users then execute the command by hitting the corresponding item with a second finger. As such, FastTap associates each command to a specific two-finger chord gesture on touchscreen. Another example is the Finger-Count menus (Bailly et al. 2012a) which improve menu navigation and item selection on tabletop. It relies on two-handed and multi-finger interaction: The number of fingers from the left hand in contact with the surface selects the menu in the menubar, while the number of fingers from the right hand selects the item within that menu. Beyond a fluid navigation in the menu system, users can quickly execute eyes-free, i.e., without the visual modality, up to 25 commands.

In conclusion, the implementation of classical methods for selecting commands varies depending on the operating system or the application. While the differences appear small at first glance, they might impact user experience. The available input and output modalities also influence the user experience. They can constrain the interaction (e.g., small screen) or offer novel opportunities (e.g., multi-touch interaction).

Designing for Command Selection

The different methods presented above are complex interaction techniques that can be decomposed in basic (and subtle) primitives that we call properties. A property improves command selection according to one or several criteria (speed, accuracy, learning, satisfaction, etc.). We already discussed some of them: does the method "permanently" use screen space, support "eyes-free" interaction, or provide a direct access to a given command? More than 100 of properties have been identified, organized, and discussed for menus (Bailly et al. 2016).

In this section, we present and analyze a large variety of visual key features of command selection. To achieve this, we first discuss the properties of a single "item," common to almost all methods, and then how these items are organized within (e.g., a single menu) and between panels (e.g., hierarchical menu).

Item

An item is the smallest component to present and execute a command.

Command Name An item (Fig. 3) often contains a command name (also called text label). When this is a hierarchical item, i.e., an item opening a menu or a palette, we favor the term category name. Choosing command or category names is challenging because users often search for functionalities (e.g., *something to keep this file on my hard drive*) rather than for known command names (e.g., "Save") (Norman and Shneiderman 1991). The general guidelines are that the names should be comprehensible and coherent to facilitate the match between the targeted functionality and the item (Norman and Shneiderman 1991). Using long labels conveys more information, but they are slower to read (or to listen in the case of audio menus) and require more screen space. Categories should guide users and encourage learning (Norman and Shneiderman 1991; Bastien and Scapin 1992; Lee and Raymond 1993). They should reflect the commands they contain while not overlapping with other categories (Norman and Shneiderman 1991).

While these guidelines are useful, they do not inform how to create a set of command names that is easy to learn and to interpret. Developers or designers can have an intuition how to name commands assuming that they are typical users, but several studies show that it is very unlikely that two people choose the same name for a given functionality (Furnas et al. 1982). It is why usability testing remains

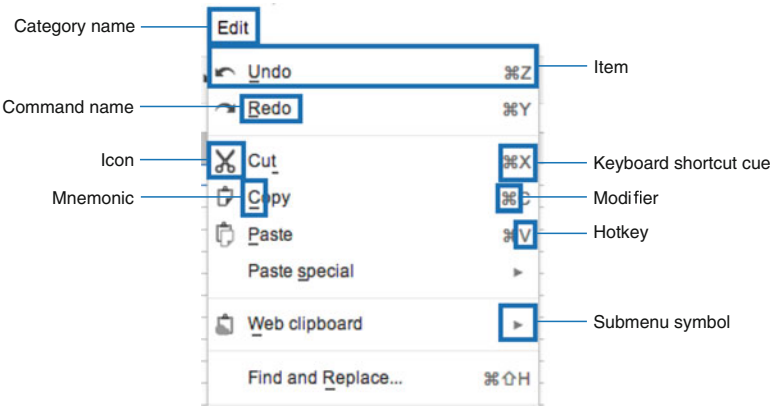

Fig. 3 The different components of a menu

critical to validate a set of command names. It is also important to involve the end users in the creation of the set of command names and menu organization (see section "Organizing Items").

Shortcut Cue Keyboard shortcut cues (Fig. 3) are generally displayed on the right side of a menu item (menu). Sometimes, they are displayed in the tooltips of the toolbar items. They are then only visible after hovering over the item for a certain time, making them difficult to discover (Malacria et al. 2013). A cheat sheet is also sometimes available. It is a panel displaying all the available keyboard shortcuts of the application or the website. However, it requires several actions (e.g., navigate in a menu) to open it and might be difficult to discover. Gesture shortcut cues are also often displayed in cheat sheets but rarely on menus (Appert and Zhai 2009) or toolbars (Bragdon et al. 2009).

Icon An icon (Fig. 3) is a pictograph representing the command. For instance, the *scissor* icon ✂ represents the command *Cut*. Icons vary along several design dimensions: shape, size, color, concreteness, complexity, distinctiveness, etc. (Lodding 1983; Ma et al. 2015; Nakamura and Zeng-Treitler 2012) which influence comprehensibility and distinguishability.

Icons have several advantages in comparison with text label. For instance, they can save space which is why items in toolbars or palettes generally only have icons, even if redundancy (text label + icon) can increase the accuracy of the selection (Wiedenbeck 1999). Moreover, the human capability of pre-attentive perceptual processing helps to rapidly locate an item whose color is already known by the user. As such, icons are more likely to support parallel search and thus to reduce the cost associated with multiple items displayed on the screen (Fleetwood and Byrne 2006). Icons also contribute to the overall aesthetic of the interface (which is a critical, yet subjective, factor of user experience (Ma et al. 2015)).

Fig. 4 Three possible strategies for embedding a shortcut cue in an icon. Left: the R is simply added on the icon. Center: the A leverages the shape of a pointer icon. Right: the E letter leverages the negative space of the power socket icon

Recently, Giannisakis et al. revealed an additional usefulness of icons: conveying shortcuts (Giannisakis et al. 2017). The advantage is to allow users who do not know or recall the (keyboard) shortcuts to easily retrieve them from the icon. This approach is especially useful when implemented in toolbars, palettes, or Ribbons as these methods permanently display icons and thus maximize shortcut exposure. Three visual strategies have been proposed to embed the shortcut cues in the icons (see Fig. 4), empty space, positive space, and negative space, so that icons can convey shortcuts without denaturing the pictograph. They also proposed to animate the icons in order to emphasize the shortcuts to different degrees.

Geometrical and Visual Attributes Designers generally consider a small number of attributes such as item position or text color (which is black or gray to indicate whether the item is enabled or disabled). However, designers can manipulate many more geometrical and visual attributes, like size, background color, transparency, etc. to improve performance. For instance, making items larger (e.g., Morphing menus (Cockburn et al. 2007)) improves motor control performance (e.g., pointing time) according to the Fitts' law (Fitts 1954). Modifying their background color (Tsandilas and Schraefel 2007) or adding icons can increase their saliency and thus improve localization time (Bailly et al. 2016). These attributes can be fixed during the interaction or changed dynamically depending on the context (see section "Adaptive Command Selection" for a discussion of the benefits and drawbacks of adaptive interfaces).

Organizing Items

Menus, palettes, Ribbons, etc. almost always leverage specific structural, semantic, and/or visual properties to present items. In this subsection, we address the question: How to organize items in a single panel? In the next section, we discuss hierarchical structure and how to navigate between these panels.

Item Organizations Items can be organized alphabetically, numerically, semantically, or in accordance with their frequency of use *within* a visual panel. When items are semantically organized, a separator (e.g., horizontal line in linear menus)

delimits the groups (also called within groups in contrast with hierarchical groups; see section "Hierarchical Structure").

Several studies compared the influence of menu organization on performance. Generally, semantic and alphabetic organizations of items are faster than unordered organizations (unordered organization is not a viable design option but serves as a baseline) (Card 1982; McDonald et al. 1983; Mehlenbacher et al. 1989). Indeed, without information about the organization, users are likely to perform a *serial (visual) search* which consists of a systematic top-to-bottom reading of items or *random search* which consists of randomly fixating items with the attempt to fixate the target one. Both of these visual search strategies are quite slow. With information about the organization, experienced users can perform a more advanced visual search by skipping some items in the menu. Typically, in semantic organizations, users can perform *foraging search*, i.e., inspecting groups likely to contain the target item but skipping the others.

The relative performance between alphabetic and semantic organization is more nuanced and depends whether the users are looking for a functionality (they do not know the exact label of the command, e.g., is it Delete, Drop, or Erase for "removing a file"?) or a command (they know the label). When searching a functionality, semantic organization is faster than alphabetic organization (McDonald et al. 1983). However, when searching for a command, the evidence is mixed: Some studies found no difference (McDonald et al. 1983), but others suggest that alphabetic is faster (Mehlenbacher et al. 1989).

Sometimes the set of items has a conventional order such as temporal order (days of the week, months of the year, most recent opened documents) or other ordinal dimension such as size (e.g., small, medium, large). The conventional order is then more appropriate than the alphabetical or semantic organization.

Finally, another possible organization is *frequency-based* organization, when command frequency is known in advance. Typically, the most-frequent items are located at the top of the menus so that they can be selected more quickly according to the Fitts' law (Fitts 1954). While sometimes effective, designers should be careful when considering this organization. Indeed, it is likely the most-frequent commands are different from one user to another.

Layout In the previous paragraphs, we implicitly assumed a *linear layout* where items are visually organized vertically or horizontally. Another common layout is the *grid layout* (Cheng and Patterson 2002), illustrated in Fig. 5 – left. One advantage is that it reduces the mean distance between items and thus reduces pointing time. Another one is that it provides more flexibility for highlighting the semantic relationships between items. For instance, related items such as "Save" and "Save As" can be located on the same row. A more recent layout is the *radial layout* (Fig. 5 – right), initially introduced with the Pie menus (Callahan et al. 1988) and popularized with the Maxis' The Sims video game series. This layout places items in a circular design at an equal radial distance from the center. This property ensures constant access time and improves global performance: Callahan et al. (1988) showed that radial menus were 15% faster than linear menus for eight items

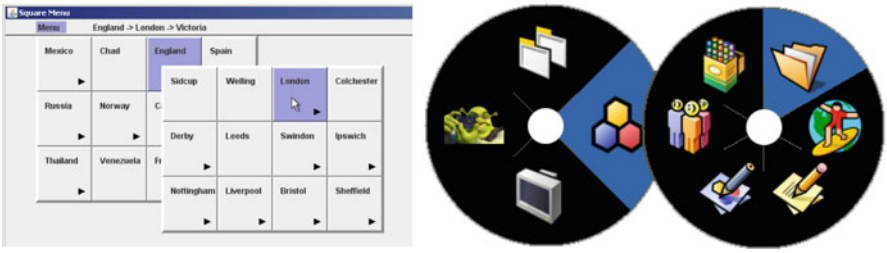

Fig. 5 Hierarchical menus using a grid layout (left) and radial (right) layout

(Callahan et al. 1988). A radial layout is also useful to strengthen semantic relations (Soliz and Paley 2003). For instance, opposite commands, "Open" and "Close," can be placed in symmetrical locations. These relations are perceived by the procedural memory, hence helping the learning and memorization of commands.

Ahlström et al. elaborated a predictive model of human performance with linear, grid, and radial layouts (Ahlström et al. 2010). Both the model and empirical data from their experiment suggest that grid layouts are faster than linear and radial layouts.

Long List Designers may have to display a large number of items in a single panel, for instance, when no hierarchical structure naturally emerges. However, some layouts, such as circular layouts, are not suitable for displaying a large number of items (Hopkins 1991). In contrast linear layouts and grid layouts better support large number of items. Examples of popular 1D long lists that can contain dozens (or hundreds) of items are contact lists, country lists, or font lists. They are generally ordered (alphabetically or historically) with only a portion of the list visible at a given time because of the limited screen space. Users can then scroll these lists, for instance, with touch-scrolling (Quinn et al. 2013) in the case of a smartphone contact list or by hovering the scrolling arrow button in the case of the History submenu of a desktop web browser. Other strategies allow to display *all* items of a long list at once. For instance, Fisheye menu (Bederson 2000) is a linear menu where only items near the cursor are shown at full size. Items that are not in the focus area are displayed at a smaller size. This visualization technique is available in the default macOS dock. A 2D variant is also available on Apple smartwatches. While Fisheye menu is attractive, empirical results suggest that grid and hierarchical menus are generally faster (Hornbæk and Hertzum 2007; Cheng and Patterson 2002).

Hierarchical Structure

There are several reasons for not displaying *all* items in a single visual panel and using a hierarchical structure instead. For instance, when a set of items grows,

designers end up being limited by the screen size, especially on small devices. Designers should then create a hierarchical structure.

Depth vs. Breadth A hierarchical structure has two main characteristics: depth and breadth. Depth is the maximum number of levels of the structure, typically the number of submenus to open before executing the commands. Breadth is the number of items per level, i.e., the number of items displayed at a given time on the current panel. One question designers can face is: Given a set of commands, what is the optimal hierarchical structure to organize them? Although many studies investigated the advantages of broad and deep structures on learning and selection performance (Snowberry et al. 1983; Kiger 1984; Larson and Czerwinski 1998; Zaphiris 2002; Cockburn and Gutwin 2009), there is no consensus on whether hierarchical structures should be deep or broad. Deep structures are necessary when there is not enough space to display all the items. They also reduce the visual complexity of the interface by hiding items less likely to be used. Users thus have less items to process as they only read those displayed in the visited submenus and skip nonrelevant submenus. However, users may have difficulties predicting from a high-level category what low-level commands fall under each of the subcategories. They can be lost in deep structures and/or use a nonoptimal pathway to reach the desired command, especially when the category names do not reflect the mental model of the user. Moreover, deep structures force users to open more visual panels, each of them requiring additional operations (e.g., key press, mouse click). In conclusion, the adequacy of a hierarchical structure depends if the user is searching for a functionality or a command, the screen constraints, the menu layout (e.g., radial layouts can contain a small number of items at given level in comparison with linear layouts), and the quality of the category names. Based on the above pros and cons of deep and broad structures, it appears that the recommendation is to generally favor breadth rather than depth to increase performance (Cockburn and Gutwin 2009).

Categories Regardless of the hierarchical structure, broad or deep, designers should decide which items are in the same categories and how these categories are named. Studies suggest that hierarchical structures generated from a group of participants (not a single participant) representative of the target users are superior than those generated by designers (e.g., Hayhoe 1990).

Several methods have been proposed to create user-generated organizations (Cooke 1994). They rely on two steps. The first step estimates the semantic distance between all commands (constructing the matrix of similarity). For instance, a group of participants estimates the relatedness of each pair of command name, but this method (pairwise relatedness ratings) is time-consuming. A less heavy method is to ask each participant to sort items into piles and then to aggregate data over participants. The second step consists of extracting groups of commands. To achieve this, one can use multidimensional scaling (MDS) to produce a 2D spatial organization that best reflects the similarity between items and group of items. Another method is hierarchical clustering analysis (HCA) to produce a tree representing each cluster and how they could be merged. At one extreme, all items

are in their own clusters, and at the other, all items are in the same cluster. The reader can find more information in Cooke (1994).

Navigation Several design factors influence how fluidly users navigate in a hierarchical structure and thus the performance and the user experience. Among them, we already discussed the choice of structure (broad or deep) as well as the quality of the category names. Another important factor is probably when and how to open the submenus. A naive implementation is to click on a hierarchical item (or parent item) to display the corresponding submenu. If the submenu does not contain the desired item, users should then perform a second click on a different submenu to continue the exploration which is time-consuming. A more advanced mechanism, called preview or previsualization (Bailly et al. 2007; Rekimoto et al. 2003), opens/closes a submenu when the cursor lies over/leaves its parent item. This facilitates visual search because users can quickly explore a set of submenus without having to perform multiple clicks. It however requires enough screen real estate to simultaneously display the current menu and the preview of the submenu. This is why this mechanism is rarely implemented on small devices such as smartphones, even if some alternative menus have been designed especially to preview submenus on small screens (Francone et al. 2009).

In conclusion, when the number of items increases, designers should consider hierarchical structure and if they favor depth vs. breadth. The general recommendation is to favor broad structures and/or user-generated structures because it is difficult for designers to generate good category names. Regardless of the hierarchical organization, the preview of the submenus facilitates the exploration and navigation.

Adaptable and Adaptive Methods for Selecting Commands

In the previous section, we highlighted *design* properties. However, some interfaces offer flexibility and modify these properties over time: the users can modify the interface themselves (adaptable method); or the system can modify the interface based on users' actions (adaptive method) (Abrahão et al. 2021).

Adaptable Command Selection

An adaptable method is a method that end users can personalize depending on their needs or preferences. The five methods presented in section "Methods for Selecting Commands" do not provide the same degree of adaptability. For instance, toolbars are highly adaptable. They often have a "customize toolbar" functionality with which users can choose which commands are displayed and how (position of the items, size of the icon and/or presence of a text label and/or separators, etc.). Keyboard shortcuts can also be adapted. Some application allows users to

modify their keyboard shortcuts directly in the applications settings, but this is not systematic. However, certain operating systems such as macOS allow users to modify existing or create keyboard shortcuts in the operating system settings. Finally, in CLIs, users can also create scripts to automate the execution of several commands at once (Thompson et al. 2007; Bland et al. 2007). In contrast, pull-down menus contain very few mechanisms to personalize this method. An exception is the "Bookmarks" menu in web navigator where users can add, remove, and manage visited web pages.

Adaptive Command Selection

The second class of methods is adaptive methods in which the system automatically modifies the content, the layout, or the style of a method depending on the user, the task, or the context. Several adaptive menus (Vanderdonckt et al. 2019) are or have been available in commercial products. The most famous one is Split menus. Split menus (Sears and Shneiderman 1994) contain two parts: a top area containing a copy of the most frequent items (generally two or three) and the bottom area containing all menu items. That way, Split menus facilitate the selection of the most frequent items by reducing the distance to reach them since they are displayed at the top of the menu. Such menus can be found, for instance, in Microsoft Office applications to select fonts. Another popular adaptive menu was the Folded menus. This menu initially displays only the most recent and frequent items reducing the number of items to read (folded). To reveal (unfold) all items, the users either hover in the menu with the mouse for a few seconds or click on the "unfold" button at the bottom of the menu. Unlike Split menus, Folded menus did not receive the same success: while they reduce selection time for high-frequency items, the cost of selecting low-frequency items is increased (Lee and Yoon 2004) making these menus too sensitive to changes in selection frequency. It results that the folded menus were removed from Microsoft Office. Finally, many menu systems contain an "Open Recent" or "History" menu where the content is automatically ordered by recency (e.g., recently opened, visited, or closed).

Ribbons also often adapt the interface. For instance, the size and organization of the icons change depending on the available screen space. Moreover, when an object of interest is selected, the tab containing the items most likely to be selected is either highlighted if already present in the bar or added after the existing ones. With the best of our knowledge, we are not aware of real-world adaptive mechanisms for keyboard and gesture shortcuts as well as command lines, probably because these methods are dedicated to expert users and generally rely on memory.

The above strategies mainly aimed to favor the selection of frequent/favorite commands. Another strategy consists in displaying progressively more items as users become more familiar with the application. For instance, "Training wheels" interfaces (Carroll and Carrithers 1984) display basic commands at first to not overwhelm the users. When users are more familiar with the application, the system provides the more advanced commands.

Challenges of adaptive methods Some adaptive methods, such as the Split menus, have been shown to improve efficiency and/or satisfy users' preferences (Sears and Shneiderman 1994; Lee and Yoon 2004). However, only a small set of careful changes can really improve usability. Indeed, the benefits of a novel (adapted) interface should be higher than the costs of the adaptations likely to occur (for instance, because the users are surprised or they do not find the command at the expected location). Typically, the cost of having to "relearn" the organization of the folded menus and the cost of the additional click might be reasons of the lack of success of this adaptive method.

Designing adaptive methods for command selection is thus challenging. Two main challenges are to determine the relative importance of each command (target policy or prediction scheme (Vanderdonckt et al. 2019)) and how to represent the corresponding items (adaptation style). The target policies often rely on heuristics to decide what to adapt. They can consider a wide variety of factors related to the user (age, preference, abilities, emotional state, previous experience, etc.), the task (current session, object of interest, etc.), the system (screen size, resolution, etc.), or the environment (location, mobility, etc.) (Abrahão et al. 2021). In practice, factors generally include item frequency, item recency, primacy, page visit duration, the previous executed command, current time and day, etc. Several heuristics combine several of these factors, and some of them also try to avoid spatial changes when not required (Fitchett and Cockburn 2012). For instance, the Microsoft Split menus determine the three most important items based on both recency and frequency (Findlater and McGrenere 2004). An emerging approach consists of developing target policies based on predictive models of user expertise and performance: given some assumptions about the user and the history (sequence of selected commands), the predictive model tends to estimate the temporal cost of an interface change at different time scale (Todi et al. 2021).

The next challenge is to determine how to represent the different items (adaptation style) given their relative importance (target policy) (Vanderdonckt et al. 2019). As discussed in section "Item", numerous geometrical (position, size, shape) and visual (background color, transparency) features can be considered to highlight some items. The main difference here is that these features evolve during the interaction depending on the target policy and can break spatial consistency which is important for performance improvement (see section "Performance Improvement When Selecting Commands"). Typically, modifying the position of items should be considered very carefully as users cannot capitalize on their previous experience. In contrast, strategies increasing the saliency of items (Findlater et al. 2009; Scarr et al. 2015) maintain spatial consistency while facilitating visual search. For instance, the Ephemeral menus (Findlater et al. 2009) first display frequent items and then make the other items visible after a delay. The user's attention is drawn to these frequent items facilitating visual search.

In summary, adaptable and adaptive methods for selecting commands have the potential to improve performance. Despite the success of some adaptive methods, they remain difficult to design as the benefits of the adaptation should overcome the cost of updating the interface.

Performance Improvement When Selecting Commands

Performance improvement with user interfaces can be categorized in four domains characterized by different opportunities to improve (Cockburn et al. 2014): intramodal improvement, intermodal improvement, vocabulary extension, and task mapping. The first three are particularly relevant regarding command selection.

Intramodal Improvement

Intramodal improvement (Cockburn et al. 2014; Malacria et al. 2013; Scarr et al. 2011) concerns performance improvement with one particular method (e.g., pointing with the mouse in the menu or using a keyboard shortcut). It can be divided into three phases (Newell and Rosenbloom 1981): the *initial performance* phase where users are globally unfamiliar with the interface, the *extended learnability* phase where users improve their performance, and the *ultimate performance* phase where users have reached an asymptotic performance with the given method. We now discuss intramodal improvement for the five methods:

Pointer-Based Intramodal Improvement (Menus and Toolbars) The *initial learning* of pointer-based methods requires users to rely on prior knowledge, visual search, and recognition to locate and activate the desired items. Once the location of the desired items is approximately known, users enter the *extended learnability* phase. Through repetition and practice, users can then reach the *ultimate performance* of pointer-based selection.

Adaptive methods (section "Adaptive Command Selection") increasing the saliency of frequent items (Findlater et al. 2009; Scarr et al. 2015) can shorten the time needed to reach the ultimate performance. This one is then plateaued by the execution time of the mechanical actions necessary to select the desired command. As such, the selection of the most frequent commands should require rapid mechanical actions to guarantee a high ultimate performance.

Shortcut-Based The initial learning of shortcut-based methods is more complicated because keyboard or gesture shortcuts are generally not readily displayed when users really need them. To use them, users first need to know the shortcuts and memorize them. As an example, imagine a user who just finished to type an e-mail and wants to activate the "send e-mail" command with its corresponding keyboard shortcut. If she does not already know the shortcut, she must browse the menubar or dwell on the corresponding toolbar button in order to display the desired keyboard shortcut cue (see shortcut cue in section "Item").

A *mnemotechnical* mapping between a command and its corresponding shortcut can shorten the period to reach the ultimate performance. For instance, using the first letter of the command as hotkey (e.g., Ctrl+P for print) or using a relevant symbolic gesture for a gestural shortcut (e.g., a star for favorite) can simplify the learning

process. However, this solution does not scale with large command sets as several command names are likely to start with the same letter (e.g., Save, Save as, Save all). That being said, shortcuts usually support a very high ultimate performance plateau by design as they do not require to navigate through a menu hierarchy.

Mnemonics Improvement With mnemonics, users progressively learn (and memorize) the key sequence corresponding to the desired command but also the respective locations of the items to select. Eventually, through practice and repetition, *ultimate performance* is reached. However, mnemonics were found slower than pointer-based selection or keyboard shortcuts in empirical studies (Malacria et al. 2013; Miller et al. 2011). One reason is that chunking a multiple-keys sequence into one single cognitive unit is difficult when using mnemonics (Miller et al. 2011).

Command Lines Improvement The initial and extended learning of CLIs is notoriously complicated: users not only need to know and memorize the commands to type, but also their associated parameters. To discover and learn CLIs, users can explore the command manual (using the *man* command) or display help facility (using the *-help* parameter with the command). Another difficulty, specific to CLIs, is that complex commands require to type a long text, which takes time, at the risk of making a typo without noticing it or noticing it only at the end when execution is requested. To improve performance when typing complex commands, users can browse the history and select previously typed commands by simply pressing the up key of the keyboard. Experienced users can also write scripts to optimize some process (see section "Adaptive Command Selection").While empirical results comparing the performance of CLIs over other methods have been mixed (Whiteside et al. 1985), they are often described as offering a high ultimate performance (Scarr et al. 2011), and frequent users of CLIs are strong advocates of this interaction method (Barrett et al. 2004).

Intermodal Improvement

Intermodal improvement concerns ways to assist users in switching to more efficient methods for executing a particular command, for instance, switching from menus to shortcuts. However, users do not always switch to these most efficient methods (Lane et al. 2005; Mackay 1991). One reason is that users may simply not be aware of the alternative method (e.g., keyboard shortcut cues displayed in menus are frequently ignored by users (Grossman et al. 2007)). A solution consists of using feedbacks after selecting an item with the mouse, for instance, by displaying a pop-up window prompting users to perform the shortcut (Krisler and Alterman 2008) or by having speech synthesis pronouncing the keyboard shortcut (Grossman et al. 2007). Another reason might be that users do not perceive the benefits of the most efficient methods (Lane et al. 2005; Odell et al. 2004; Tak et al. 2013). This problem can be alleviated by encouraging users to reflect about their performance

through skillometers, graphical components that provide information about users' ongoing performance, in order to assist them in quantifying the associated costs and benefits of switching to alternative methods (Malacria et al. 2013). Finally, Carroll's "paradox of the active user" (Carroll and Rosson 1987) suggests that users might be simply too engaged in their tasks to consider learning alternative strategies or methods, or do not switch because they "satisfice" (Simon 1966) with the method they usually use. Laboratory experiments showed unsurprisingly that radical approaches forcing users to use the most efficient method or penalizing them for not doing it favor the switch, but these approaches might be too constraining to be acceptable in practice (Grossman et al. 2007; Krisler and Alterman 2008). More subtle approaches, such as calm notifications, increase the visibility of the more efficient methods without demanding too much attention and may be more acceptable (Scarr et al. 2011).

Assuming that users transition, the intermodal expertise framework characterizes this transition (Scarr et al. 2011) (Fig. 6). It postulates that users are likely to suffer a temporary, yet substantial, performance dip when switching to the second modality.

In the context of keyboard shortcuts, this performance dip often arises because of the necessity to recall the mapping with the associated command, resulting in longer executing time or more errors than when using the slower, but "easier," menu method. Several solutions have been proposed in addition to using a mnemonical mapping as described above, for instance, displaying the keyboard shortcut cues on the screen when a modifier key is pressed, either as an overlay on toolbar

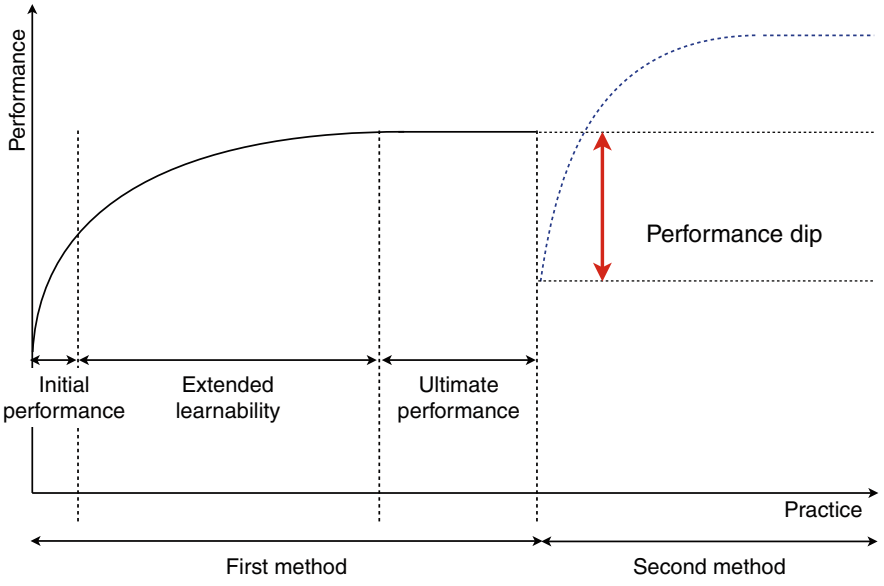

Fig. 6 Characterization of the intermodal transition from one method (e.g., menu) to a second one more efficient (e.g., shortcuts)

buttons (Malacria et al. 2013) or as a list of commands available in the menus (Malacria et al. 2013; Tak et al. 2013), or by using an on-screen visual keyboard (Lewis et al. 2020). These solutions remove the necessity to memorize the shortcuts beforehand, increase the usage of the shortcuts, and lower error rates during laboratory experiments. These positive results can explain that such solutions can be found in commercial applications or operating systems, such as in ChromeOS, CheatSheet, or the Slack desktop application.

On the other hand, some interaction techniques were designed with this transition in mind, such as Marking menus (Fig. 2) mentioned in section "The Diversity of Methods". In order to minimize this performance dip, they implement the principle of rehearsal (Kurtenbach and Buxton 1991), that is, that *"guidance should be a physical rehearsal of the way an expert would issue a command"*. Since users perform similar movements when executing the commands from the menu or when performing the gesture without the menu to appear, it results that users *rehearse* the gesture shortcuts by repeatedly selecting commands in the menus. However, recent research suggests that Marking menus would also work if the menu appeared systematically and instantly, thus removing the need to memorize the gestures and switch between modes (Henderson et al. 2020).

Vocabulary Extension

Vocabulary extension concerns ways to assist users to broaden their knowledge of the range of commands available. Indeed, users sometimes are not aware of the whole vocabulary of commands to efficiently achieving a task (Matejka et al. 2009; Pong and Malacria 2019). As an example, a user may create a copy of a file by using the "copy" and "paste" commands sequentially because she is not aware that the "duplicate" command exists. Similarly, in a photo editing software, she may paint filled circles over eyes' pupils in order to remove the red eyes from photographs because she is not aware that a dedicated "red-eye removal" command exists.

Several approaches have been proposed to extend the vocabulary of commands users are aware of. One of them consists in exposing to users commands that might be of their interest. For instance, "tip of the day" (Fig. 7) or onboarding interfaces (Crumlish and Malone 2009) (interfaces helping users to get started with an application) illustrate random or novel commands to users. However, these interfaces can be perceived as unwanted distraction, especially because the information provided is usually disconnected with user's own task. In contrast, ambient recommendations provide contextual assistive content (Ekstrand et al. 2011; Matejka et al. 2011). For instance, "Patina" overlays toolbar buttons with a colored heatmap to emphasize command recommendations without impeding access to them (Matejka et al. 2013). Such methods require algorithms that recommend commands both useful and novel for the user. Previous research suggests that algorithms exploiting both social and task-based data would generate the highest number of useful recommendations (Wiebe et al. 2016; Li et al. 2011). Another approach consists of illustrating the effect of a command in order to assist users in choosing the correct one. Side

Fig. 7 Example of a "tip of the day" in the Konsole app that explains how to activate a novel command

Views (Terry and Mynatt 2002) or ToolClips (Grossman and Fitzmaurice 2010) preview the effect of a command using text or a brief animation, respectively. Finally, Lafreniere et al. presented design concepts combining both command preview and recommendation (Lafreniere et al. 2015).

Summary and Future Directions

Summary

Command selection is a fundamental and frequent task in human-computer interaction concerning all interactive systems. It has also been a vibrant object of research for decades, as it involves numerous phenomena such as visual search, pointing, skill acquisition, or decision-making. We now revisit our four main research questions:

What is command selection and what are the main methods for selecting commands?

While there is no general agreement on the definition and the different methods for selecting commands, we proposed to define command selection as the task that consists of choosing one specific command among a set of commands. We also organized the diverse interaction techniques to select commands into five classes: command line interfaces (CLIs), menus, mnemonics, toolbars, and shortcuts, which generally coexist in interactive systems so that users can freely choose between them based on users' preferences, goals, or tasks. These methods have key differences regarding their input modality (mouse vs. keyboard), the amount of screen space

required, how commands are accessed (direct vs. hierarchical), the capacity to let users select commands eyes-free, or the need to rely on explicit recall in order to use them efficiently.

What are the design factors of command selection?

The design space of command selection is enormous. Typically, designers should consider three main levels of granularity. First is item, which is the smallest component to present and execute a command. An item can contain the name of a command, an icon, a shortcut cue, and additional geometrical and visual attributes (e.g., background color).

Second, designers should decide how to organize items in a single panel. They can organize items alphabetically, semantically, or by frequency. A semantic organization is generally more efficient when users do not know the exact name of the command. An alphabetical one is preferred when users know the exact command name or when the list of items is extremely large. Frequency-based organizations are rarely recommended. The visual structure (or layout) is also important. While linear and grid layouts are common in GUIs, the circular layout is an interesting alternative when the number of items is relatively small.

Third, designers will often have to choose a hierarchical structure to further organize items, especially when the number of items is large or the screen is small. The recommendation is to favor a broad structure over a deep one. Moreover, designers should be very careful about the names of the categories as they determine how easily users navigate and find the desired command.

These design choices depend on multiple factors. Among them, the devices, operating systems or applications the commands will be selected in. For instance, Windows and Apple have their own design guidelines regarding the choice of the icons or keyboard shortcuts. Similarly, a device having multi-touch capability or a small screen provides opportunities or constraints to consider when designing methods for selecting commands.

How to adapt command selection to the user?

Two main groups of methods exist for adapting command selection to a specific user: adaptable and adaptive methods. Adaptable methods allow end users to personalize the methods depending on their needs or preferences. In contrast, adaptive methods automatically adapt the interface depending on the context. Adaptive methods are promising but should be considered very carefully as the benefits do not necessarily overcome the cost of the interface changes. It is generally recommended to design methods which maintain spatial consistency to facilitate visual search.

How to improve command selection performance?

By considering the whole design space of command selection, designers have three main opportunities to improve command selection. First, they can improve the performance of a given method (intramodal improvement), e.g., menus, regardless of whether users may use alternative methods. Second, when several methods are available for selecting commands, they can work on intermodal improvement by assisting users in switching to a more efficient method. The last opportunity of

improvement, independent of the method, consists in extending users' vocabulary of command selection, to broaden their knowledge of available commands.

Future Work

Despite decades of research on command selection, several directions for future work remain. In this section, we focus on three of them.

Vocabulary extension As illustrated in section "Performance Improvement When Selecting Commands", significant research has been conducted on intramodal and intermodal improvement. In contrast, vocabulary extension was not studied to the same extent. However, modern applications provide an increasing (sometimes very large) number of commands. Therefore, users are likely to miss some of them. This problem is exacerbated with small screens, which encourage designers to hide commands (e.g., by using deeper hierarchical structures) forcing users to discover them (Pong and Malacria 2019).

The challenge is to provide guidelines and develop methods to foster vocabulary extension. The methods should go beyond recommending systems. For instance, they should determine why some commands are not used. Maybe users really do not need them, or maybe it is because they are not aware of them or cannot find them. This will require a better understanding on how users spontaneously discover commands and how they extend their vocabulary as well as the human and design factors influencing vocabulary extension.

Designing for emerging technologies Interactive systems are continuously evolving with novel input and output modalities. These modalities introduce both opportunities and constraints for command selection (section "Input and Output Modalities"). They also tend to suggest that it is necessary to completely revisit how users select commands on these systems. However, several tasks (discovering, exploring, navigating, activating) and concepts (screen space, hierarchical structure, direct access) remain regardless of the platform, be it a desktop computer or a smartphone.

The challenge is to develop methodologies to transpose findings from extensively studied environments (desktop and smartphone) to emerging interactive technologies such as virtual reality (Park et al. 2019; Dachselt and Hübner 2006; Fennedy et al. 2021), augmented reality (Saidi et al. 2021), wearable computing (Bailly et al. 2012b), or body-based interaction (Fruchard et al. 2018; Weigel et al. 2014; Harrison et al. 2010). It requires to refine our understanding of the foundations of command section, i.e., what are precisely the similarities and the differences among interactive systems regarding command selection. This is necessary, for instance, to anticipate and generalize the discoverability problems of pointer-based or touch-based interaction to new input modalities such as hand postures (Liu et al. 2015), voice-based control (Yang et al. 2020), or gaze and micro-gestures (Wambecke et al. 2021). This is especially important as some commands can only be activated once

the corresponding modality has been discovered (e.g., executing the Undo command by shaking an iOS device (Pogue 2012)).

Computational models Another direction is the elaboration of computational models. Many empirical studies have been conducted and various interaction techniques have been designed. In comparison, few computational models of command selection have been elaborated to encapsulate this scientific knowledge. Moreover, these models mainly focus on one method, menu, and on a small number of properties such as item position (Byrne 2001; Lee and Macgregor 1985), even if some models of command selection also consider practice (Cockburn et al. 2007; Bailly et al. 2014), semantics (Chen et al. 2015; Dayama et al. 2021), saliency (Chen et al. 2015), stability of the interface (or learnability) (Cockburn et al. 2007), a different layout (Pfeuffer and Li 2018), or gesture shortcuts (Cao and Zhai 2007).

The challenge is to develop models covering the different methods, their properties, the interaction between these properties, as well as the transition between these methods (Bailly et al. 2021). These models should be able to provide a preliminary evaluation of a specific design. They should also be introduced in optimization methods (Dayama et al. 2021; Bailly et al. 2013a; Matsui and Yamada 2008) to explore in a systematic way the design space of command selection.

References

Abrahão S, Insfran E, Sluÿters A, Vanderdonckt J (2021) Model-based intelligent user interface adaptation: challenges and future directions. Softw Syst Model 20:1–15

Accot J, Zhai S (2002) More than dotting the i's – foundations for crossing-based interfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'02, New York. Association for Computing Machinery, pp 73–80

Ahlström D, Cockburn A, Gutwin C, Irani P (2010) Why it's quick to be square: modelling new and existing hierarchical menu designs. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'10, New York. Association for Computing Machinery, pp 1371–1380

Apitz G, Guimbretière F (2004) Crossy: a crossing-based drawing application. In: Proceedings of the 17th annual ACM symposium on user interface software and technology, UIST'04, New York. Association for Computing Machinery, pp 3–12

Appert C, Zhai S (2009) Using strokes as command shortcuts: cognitive benefits and toolkit support. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'09, New York. Association for Computing Machinery, pp 2289–2298

Bailly G, Lecolinet E, Nigay L (2007) Wave menus: improving the novice mode of hierarchical marking menus. In: Proceedings of the 11th IFIP TC 13 international conference on human-computer interaction, INTERACT'07, Berlin/Heidelberg. Springer, pp 475–488

Bailly G, Demeure A, Lecolinet E, Nigay L (2008) Multitouch menu (MTM). In: Proceedings of the 20th conference on l'interaction homme-machine, IHM'08, New York. Association for Computing Machinery, pp 165–168

Bailly G, MüLler J, Lecolinet E (2012a) Design and evaluation of finger-count interaction: combining multitouch gestures and menus. Int J Hum-Comput Stud 70(10):673–689

Bailly G, Müller J, Rohs M, Wigdor D, Kratz S (2012b) Shoesense: a new perspective on gestural interaction and wearable applications. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'12, New York. Association for Computing Machinery, pp 1239–1248

Bailly G, Oulasvirta A, Kötzing T, Hoppe S (2013a) Menuoptimizer: interactive optimization of menu systems. In: Proceedings of the 26th annual ACM symposium on user interface software and technology, UIST'13, New York. Association for Computing Machinery, pp 331–342

Bailly G, Pietrzak T, Deber J, Wigdor DJ (2013b) Métamorphe: augmenting hotkey usage with actuated keys. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'13, New York. Association for Computing Machinery, pp 563–572

Bailly G, Oulasvirta A, Brumby DP, Howes A (2014) Model of visual search and selection time in linear menus. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'14, New York. Association for Computing Machinery, pp 3865–3874

Bailly G, Lecolinet E, Nigay L (2016) Visual menu techniques. ACM Comput Surv 49(4):1–41

Bailly G, Khamassi M, Girard B (2021) Computational Model of the Transition from Novice to Expert Interaction Techniques. ACM Trans. Comput.-Hum. Interact. Just Accepted (December 2021). https://doi.org/10.1145/3505557

Barrett R, Kandogan E, Maglio PP, Haber EM, Takayama LA, Prabaker M (2004) Field studies of computer system administrators: analysis of system management tools and practices. In: Proceedings of the 2004 ACM conference on computer supported cooperative work, CSCW'04, New York. Association for Computing Machinery, pp 388–395

Bastien JMC, Scapin DL (1992) A validation of ergonomic criteria for the evaluation of human-computer interfaces. Int J Hum-Comput Interact 4(2):183–196

Bederson BB (2000) Fisheye menus. In: Proceedings of the 13th annual ACM symposium on user interface software and technology, UIST'00, New York. Association for Computing Machinery, pp 217–225

Bland W, Naughton T, Vallee G, Scott SL (2007) Design and implementation of a menu based oscar command line interface. In: 21st international symposium on high performance computing systems and applications (HPCS'07), p 25

Bragdon A, Zeleznik R, Williamson B, Miller T, LaViola JJ (2009) Gesturebar: improving the approachability of gesture-based interfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'09, New York. Association for Computing Machinery, pp 2269–2278

Buschek D, Roppelt B, Alt F (2018) Extending keyboard shortcuts with arm and wrist rotation gestures. Association for Computing Machinery, New York, pp 1–12

Byrne MD (2001) ACT-R/PM and menu selection. Int J Hum-Comput Stud 55(1):41–84

Callahan J, Hopkins D, Weiser M, Shneiderman B (1988) An empirical comparison of pie vs. linear menus. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'88, New York. Association for Computing Machinery, pp 95–100

Cao X, Zhai S (2007) Modeling human performance of pen stroke gestures. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'07, New York. Association for Computing Machinery, pp 1495–1504

Card SK (1982) User perceptual mechanisms in the search of computer command menus. In: Proceedings of the 1982 conference on human factors in computing systems, CHI'82, New York. Association for Computing Machinery, pp 190–196

Carroll JM, Carrithers C (1984) Training wheels in a user interface. Commun ACM 27(8):800–806

Carroll JM, Rosson MB (1987) Paradox of the active user. In: Interfacing thought: cognitive aspects of human-computer interaction, MIT Press, Cambridge, pp 80–111

Casadei V, Granollers T, Zaina L (2017) Investigating accessibility issues of UI mobile design patterns in online communities: a virtual ethnographic study. In: Proceedings of the XVI Brazilian symposium on human factors in computing systems, IHC 2017, New York. Association for Computing Machinery

Chen X, Bailly G, Brumby DP, Oulasvirta A, Howes A (2015) The emergence of interactive behavior: a model of rational menu search. In: Proceedings of the 33rd annual ACM conference on human factors in computing systems, CHI'15, New York. Association for Computing Machinery, pp 4217–4226

Cheng H-I, Patterson PE (2002) The grid menu: efficient and robust selection of menu-items. Proc Hum Factors Ergon Soc Annu Meet 46(14):1281–1285

Cockburn A, Gutwin C (2009) A predictive model of human performance with scrolling and hierarchical lists. Hum–Comput Interact 24(3):273–314

Cockburn A, Gutwin C, Greenberg S (2007) A predictive model of menu performance. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'07, New York. Association for Computing Machinery, pp 627–636

Cockburn A, Gutwin C, Scarr J, Malacria S (2014) Supporting novice to expert transitions in user interfaces. ACM Comput Surv 47(2):1–36

Cooke NJ (1994) Varieties of knowledge elicitation techniques. Int J Hum-Comput Stud 41(6):801–849

Crumlish C, Malone E (2009) Designing social interfaces: principles, patterns, and practices for improving the user experience. O'Reilly Media, Inc., Beijing

Dachselt R, Hübner A (2006) A survey and taxonomy of 3D menu techniques. In: Proceedings of the 12th eurographics conference on virtual environments, EGVE'06, Goslar. Eurographics Association, pp 89–99

Dayama NR, Shiripour M, Oulasvirta A, Ivanko E, Karrenbauer A (2021) Foraging-based optimization of menu systems. Int J Hum-Comput Stud 151:102624

Dubois E, Serrano M, Raynal M (2018) Rolling-menu: rapid command selection in toolbars using roll gestures with a multi-DoF mouse. Association for Computing Machinery, New York, pp 1–12

Ekstrand M, Li W, Grossman T, Matejka J, Fitzmaurice G (2011) Searching for software learning resources using application context. In: Proceedings of the 24th annual ACM symposium on user interface software and technology, UIST'11, New York. Association for Computing Machinery, pp 195–204

Fennedy K, Malacria S, Lee H, Perrault ST (2020) Investigating performance and usage of input methods for soft keyboard hotkeys. In: 22nd international conference on human-computer interaction with mobile devices and services, MobileHCI'20, New York. Association for Computing Machinery

Fennedy K, Hartmann J, Roy Q, Perrault ST, Vogel D (2021) Octopocus in VR: using a dynamic guide for 3D mid-air gestures in virtual reality. IEEE Trans Vis Comput Graph 27(12):4425–4438

Findlater L, McGrenere J (2004) A comparison of static, adaptive, and adaptable menus. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'04, New York. Association for Computing Machinery, pp 89–96

Findlater L, Moffatt K, McGrenere J, Dawson J (2009) Ephemeral adaptation: the use of gradual onset to improve menu selection performance. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'09, New York. Association for Computing Machinery, pp 1655–1664

Fitchett S, Cockburn A (2012) Accessrank: predicting what users will do next. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'12, New York. Association for Computing Machinery, pp 2239–2242

Fitts PM (1954) The information capacity of the human motor system in controlling the amplitude of movement. J Exp Psychol 47(6):381

Fleetwood MD, Byrne MD (2006) Modeling the visual search of displays: a revised act-r model of icon search based on eye-tracking data. Hum-Comput Interact 21(2):153–197

Francone J, Bailly G, Nigay L, Lecolinet E (2009) Wavelet menus: a stacking metaphor for adapting marking menus to mobile devices. In: Proceedings of the 11th international conference on human-computer interaction with mobile devices and services, MobileHCI'09, New York. Association for Computing Machinery

Fruchard B, Lecolinet E, Chapuis O (2018) Impact of semantic aids on command memorization for on-body interaction and directional gestures. In: Proceedings of the 2018 international conference on advanced visual interfaces, AVI'18, New York. Association for Computing Machinery

Fruchard B, Lecolinet E, Chapuis O (2020) Side-crossing menus: enabling large sets of gestures for small surfaces. Proc ACM Hum-Comput Interact 4(ISS):1–19

Furnas GW, Gomez LM, Landauer TK, Dumais ST (1982) Statistical semantics: how can a computer use what people name things to guess what things people mean when they name things? In: Proceedings of the 1982 conference on human factors in computing systems, pp 251–253

Ghomi E, Huot S, Bau O, Beaudouin-Lafon M, Mackay WE (2013) Arpège: learning multitouch chord gestures vocabularies. In: Proceedings of the 2013 ACM international conference on interactive tabletops and surfaces, ITS'13, New York. Association for Computing Machinery, pp 209–218

Giannisakis E, Bailly G, Malacria S, Chevalier F (2017) Iconhk: using toolbar button icons to communicate keyboard shortcuts. In: Proceedings of the 2017 CHI conference on human factors in computing systems, CHI'17, New York. Association for Computing Machinery, pp 4715–4726

Goguey A, Malacria S, Cockburn A, Gutwin C (2019) Reducing error aversion to support novice-to-expert transitions with fasttap. In: Proceedings of the 31st conference on l'interaction homme-machine, IHM'19, New York. Association for Computing Machinery

Grossman T, Fitzmaurice G (2010) Toolclips: an investigation of contextual video assistance for functionality understanding. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'10, New York. Association for Computing Machinery, pp 1515–1524

Grossman T, Dragicevic P, Balakrishnan R (2007) Strategies for accelerating on-line learning of hotkeys. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'07, New York. Association for Computing Machinery, pp 1591–1600

Guimbretiére F, Winograd T (2000) Flowmenu: combining command, text, and data entry. In: Proceedings of the 13th annual ACM symposium on user interface software and technology, UIST'00, New York. Association for Computing Machinery, pp 213–216

Gutwin C, Cockburn A, Scarr J, Malacria S, Olson SC (2014) Faster command selection on tablets with fasttap. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'14, New York. Association for Computing Machinery, pp 2617–2626

Harrison C, Tan D, Morris D (2010) Skinput: appropriating the body as an input surface. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'10, New York. Association for Computing Machinery, pp 453–462

Hascoët M, Collomb M, Cance J (2006) Accelerating object-command transitions with pie menus. In: ENACTIVE, Montpellier, pp 109–111

Hayhoe D (1990) Sorting-based menu categories. Int J Man-Mach Stud 33(6):677–705

Henderson J, Malacria S, Nancel M, Lank E (2020) Investigating the necessity of delay in marking menu invocation. In: Proceedings of the 2020 CHI conference on human factors in computing systems, CHI'20, New York. Association for Computing Machinery, pp 1–13

Hendy J, Booth KS, McGrenere J (2010) Graphically enhanced keyboard accelerators for GUIs. In: Proceedings of graphics interface 2010, GI'10, CAN 2010. Canadian Information Processing Society, pp 3–10

Hopkins D (1991) The design and implementation of pie menus. Dr. Dobb's J 16(12):16–26

Hornbæk K, Hertzum M (2007) Untangling the usability of fisheye menus. ACM Trans Comput-Hum Interact 14(2):6–es

Keddisseh E, Serrano M, Dubois E (2021) KeyTch: combining the keyboard with a touchscreen for rapid command selection on toolbars. Association for Computing Machinery, New York

Kiger JI (1984) The depth/breadth trade-off in the design of menu-driven user interfaces. Int J Man-Mach Stud 20(2):201–213

Krisler B, Alterman R (2008) Training towards mastery: overcoming the active user paradox. In: Proceedings of the 5th Nordic conference on human-computer interaction: building bridges, NordiCHI'08, New York. Association for Computing Machinery, pp 239–248

Kurtenbach GP (1993) The design and evaluation of marking menus. PhD thesis, CAN. UMI Order No. GAXNN-82896

Kurtenbach G, Buxton W (1991) Issues in combining marking and direct manipulation techniques. In: Proceedings of the 4th annual ACM symposium on user interface software and technology, UIST'91, New York. Association for Computing Machinery, pp 137–144

Kurtenbach G, Buxton W (1994) User learning and performance with marking menus. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'94, New York. Association for Computing Machinery, pp 258–264

Kurtenbach G, Fitzmaurice GW, Owen RN, Baudel T (1999) The hotbox: efficient access to a large number of menu-items. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'99, New York. Association for Computing Machinery, pp 231–237

Lafreniere B, Chilana PK, Fourney A, Terry MA (2015) These aren't the commands you're looking for: addressing false feedforward in feature-rich software. In: Proceedings of the 28th annual ACM symposium on user interface software & technology, UIST'15, New York. Association for Computing Machinery, pp 619–628

Lafreniere B, Gutwin C, Cockburn A (2017) Investigating the post-training persistence of expert interaction techniques. ACM Trans Comput-Hum Interact 24(4):1–46

Lane DM, Napier HA, Peres SC, Sandor A (2005) Hidden costs of graphical user interfaces: failure to make the transition from menus and icon toolbars to keyboard shortcuts. Int J Hum-Comput Interact 18(2):133–144

Larson K, Czerwinski M (1998) Web page design: implications of memory, structure and scent for information retrieval. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'98. ACM Press/Addison-Wesley Publishing Co, pp 25–32

Lazzaro N (2009) Why we play: affect and the fun of games. Hum-Comput Interact Des Divers Users Domains 155:679–700

Lee E, Macgregor J (1985) Minimizing user search time in menu retrieval systems. Hum Factors 27(2):157–162

Lee ES, Raymond DR (1993) Menu-driven systems. Encycl Microcomput 11:101–127

Lee D-S, Yoon WC (2004) Quantitative results assessing design issues of selection-supportive menus. Int J Ind Ergon 33(1):41–52

Lepinski GJ, Grossman T, Fitzmaurice G (2010) The design and evaluation of multitouch marking menus. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'10, New York. Association for Computing Machinery, pp 2233–2242

Lewis B, d'Eon G, Cockburn A, Vogel D (2020) Keymap: improving keyboard shortcut vocabulary using Norman's mapping. In: Proceedings of the 2020 CHI conference on human factors in computing systems, CHI'20, New York. Association for Computing Machinery, pp 1–10

Li W, Matejka J, Grossman T, Konstan JA, Fitzmaurice G (2011) Design and evaluation of a command recommendation system for software applications. ACM Trans Comput-Hum Interact 18(2):1–35

Liu M, Nancel M, Vogel D (2015) Gunslinger: subtle arms-down mid-air interaction. In: Proceedings of the 28th annual ACM symposium on user interface software & technology, UIST'15, New York. Association for Computing Machinery, pp 63–71

Lodding KN (1983) Iconic interfacing. IEEE Comput Graph Appl 3(02):11–20

Mackay WE (1991) Triggers and barriers to customizing software. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'91, New York. Association for Computing Machinery, pp 153–160

Ma X, Matta N, Cahier J-P, Qin C, Cheng Y (2015) From action icon to knowledge icon: objective-oriented icon taxonomy in computer science. Displays 39:68–79

Malacria S, Bailly G, Harrison J, Cockburn A, Gutwin C (2013) Promoting hotkey use through rehearsal with exposehk. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'13, New York. Association for Computing Machinery, pp 573–582

Malacria S, Scarr J, Cockburn A, Gutwin C, Grossman T (2013) Skillometers: reflective widgets that motivate and help users to improve performance. In: Proceedings of the 26th annual ACM symposium on user interface software and technology, UIST'13, New York. Association for Computing Machinery, pp 321–330

Matejka J, Li W, Grossman T, Fitzmaurice G (2009) Communitycommands: command recommendations for software applications. In: Proceedings of the 22nd annual ACM symposium on user interface software and technology, UIST'09, New York. Association for Computing Machinery, pp 193–202

Matejka J, Grossman T, Fitzmaurice G (2011) Ambient help. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'11, New York. Association for Computing Machinery, pp 2751–2760

Matejka J, Grossman T, Fitzmaurice G (2013) Patina: dynamic heatmaps for visualizing application usage. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'13, New York. Association for Computing Machinery, pp 3227–3236

Matsui S, Yamada S (2008) Optimizing hierarchical menus by genetic algorithm and simulated annealing. In: Proceedings of the 10th annual conference on genetic and evolutionary computation, GECCO'08, New York. Association for Computing Machinery, pp 1587–1594

McDonald JE, Stone JD, Liebelt LS (1983) Searching for items in menus: the effects of organization and type of target. In: Proceedings of the human factors society annual meeting, vol 27. SAGE Publications, Los Angeles, pp 834–837

Mehlenbacher B, Duffy TM, Palmer J (1989) Finding information on a menu: linking menu organization to the user's goals. Hum-Comput Interact 4(3):231–251

Miller CS, Denkov S, Omanson RC (2011) Categorization costs for hierarchical keyboard commands. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'11, New York. Association for Computing Machinery, pp 2765–2768

Murillo SR, Sánchez JA (2014) Empowering interfaces for system administrators: keeping the command line in mind when designing GUIs. In: Proceedings of the XV international conference on human computer interaction, Interacción'14, New York. Association for Computing Machinery

Nakamura C, Zeng-Treitler Q (2012) A taxonomy of representation strategies in iconic communication. Int J Hum-Comput Stud 70(8):535–551

Newell A, Rosenbloom PS (1981) Mechanisms of skill acquisition and the law of practice. Cogn Skills Acquis 1(1981):1–55

Norman KL, Shneiderman B (1991) The psychology of menu selection: designing cognitive control at the human/computer interface. Greenwood Publishing Group Inc.

Odell DL, Davis RC, Smith A, Wright PK (2004) Toolglasses, marking menus, and hotkeys: a comparison of one and two-handed command selection techniques. In: Proceedings of graphics interface 2004, GI'04, Waterloo. Canadian Human-Computer Communications Society, pp 17–24

Park C, Cho H, Park S, Yoon Y-S, Jung S-U (2019) Handposemenu: hand posture-based virtual menus for changing interaction mode in 3D space. In: Proceedings of the 2019 ACM international conference on interactive surfaces and spaces, ISS'19, New York. Association for Computing Machinery, pp 361–366

Pfeuffer K, Li Y (2018) Analysis and modeling of grid performance on touchscreen mobile devices. Association for Computing Machinery, New York, pp 1–12

Pogue D (2012) iPhone: the missing manual. O'Reilly Media, paperback edition, 11

Pong NKC, Malacria S (2019) Awareness, usage and discovery of swipe-revealed hidden widgets in iOS. In: Proceedings of the 2019 ACM international conference on interactive surfaces and spaces, ISS'19, New York. Association for Computing Machinery, pp 193–204

Pook S, Lecolinet E, Vaysseix G, Barillot E (2000) Control menus: excecution and control in a single interactor. In: CHI'00 extended abstracts on human factors in computing systems, CHI EA'00, New York. Association for Computing Machinery, pp 263–264

Quinn P, Malacria S, Cockburn A (2013) Touch scrolling transfer functions. In: Proceedings of the 26th annual ACM symposium on user interface software and technology, UIST'13, New York. Association for Computing Machinery, pp 61–70

Rekimoto J, Ishizawa T, Schwesig C, Oba H (2003) Presense: interaction techniques for finger sensing input devices. In: Proceedings of the 16th annual ACM symposium on user interface software and technology, UIST'03, New York. Association for Computing Machinery, pp 203–212

Saidi H, Dubois E, Serrano M (2021) HoloBar: rapid command execution for head-worn AR exploiting around the field-of-view interaction. Association for Computing Machinery, New York

Sampath H, Merrick A, Macvean A (2021) Accessibility of command line interfaces. Association for Computing Machinery, New York

Scarr J, Cockburn A, Gutwin C, Quinn P (2011) Dips and ceilings: understanding and supporting transitions to expertise in user interfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'11, New York. Association for Computing Machinery, pp 2741–2750

Scarr J, Gutwin C, Cockburn A, Bunt A (2015) Stencilmaps and ephemeralmaps: spatially stable interfaces that highlight command subsets. Behav Inf Technol 34(11):1092–1106

Sears A, Shneiderman B (1994) Split menus: effectively using selection frequency to organize menus. ACM Trans Comput-Hum Interact 1(1):27–51

Shneiderman B, Plaisant C, Cohen MS, Jacobs S, Elmqvist N, Diakopoulos N (2016) Designing the user interface: strategies for effective human-computer interaction. Pearson, Hoboken

Simon HA (1966) Theories of decision-making in economics and behavioural science. Palgrave Macmillan London, pp 1–28

Snowberry K, Parkinson SR, Sisson N (1983) Computer display menus. Ergonomics 26(7):699–712

Soliz E, Paley WB (2003) A re-interpretation of marking menus: the usage of gestalt principles as cognitive tools. ACM UIST'03, poster

Tak S, Westendorp P, van Rooij I (2013) Satisficing and the use of keyboard shortcuts: being good enough is enough? Interact Comput 25(5):404–416

Terry M, Mynatt ED (2002) Side views: persistent, on-demand previews for open-ended tasks. In: Proceedings of the 15th annual ACM symposium on user interface software and technology, UIST'02, New York. Association for Computing Machinery, pp 71–80

Thompson RS, Rantanen EM, Yurcik W, Bailey BP (2007) Command line or pretty lines? Comparing textual and visual interfaces for intrusion detection. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'07, New York. Association for Computing Machinery, p 1205

Todi K, Bailly G, Leiva L, Oulasvirta A (2021) Adapting user interfaces with model-based reinforcement learning. Association for Computing Machinery, New York

Tsandilas T, Schraefel MC (2007) Bubbling menus: a selective mechanism for accessing hierarchical drop-down menus. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'07, New York. Association for Computing Machinery, pp 1195–1204

Tucker AB (2004) Computer science handbook. CRC Press, Boca Raton

Vanderdonckt J, Bouzit S, Calvary G, Chêne D (2019) Exploring a design space of graphical adaptive menus: normal vs. small screens. ACM Trans Interact Intell Syst 10(1):1–40

Wagner J, Lecolinet E, Selker T (2014) Multi-finger chords for hand-held tablets: recognizable and memorable. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'14, New York. Association for Computing Machinery, pp 2883–2892

Walter R, Bailly G, Müller J (2013) Strikeapose: revealing mid-air gestures on public displays. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'13, New York. Association for Computing Machinery, pp 841–850

Wambecke J, Goguey A, Nigay L, Dargent L, Hauret D, Lafon S, de Visme J-SL (2021) M[eye]cro: eye-gaze+microgestures for multitasking and interruptions. Proc ACM Hum-Comput Interact 5(EICS):1–22

Weigel M, Mehta V, Steimle J (2014) More than touch: understanding how people use skin as an input surface for mobile computing. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'14, New York. Association for Computing Machinery, pp 179–188

Whiteside J, Jones S, Levy PS, Wixon D (1985) User performance with command, menu, and iconic interfaces. In: Proceedings of the SIGCHI conference on human factors in computing systems, CHI'85, New York. Association for Computing Machinery, pp 185–191

Wiebe M, Geiskkovitch DY, Bunt A (2016) Exploring user attitudes towards different approaches to command recommendation in feature-rich software. In: Proceedings of the 21st international conference on intelligent user interfaces, IUI'16, New York. Association for Computing Machinery, pp 43–47

Wiedenbeck S (1999) The use of icons and labels in an end user application program: an empirical study of learning and retention. Behav Inf Technol 18(2):68–82

Yang J(J), Lam MS, Landay JA (2020) Dothishere: multimodal interaction to improve cross-application tasks on mobile devices. In: Proceedings of the 33rd annual ACM symposium on user interface software and technology, UIST'20, New York. Association for Computing Machinery, pp 35–44

Zaphiris P (2002) Age differences and the depth-breadth tradeoff in hierarchical online information systems. PhD thesis, AAI3047597

Zheng J, Vogel D (2016) Finger-aware shortcuts. In: Proceedings of the 2016 CHI conference on human factors in computing systems, CHI'16, New York. Association for Computing Machinery, pp 4274–4285