Stripes

A Conceptual Operating System User Interface

G. Martin R. Gimpl

University of Art and Design Helsinki

Media Lab Helsinki

Master's Thesis

10/16/08

Department	*Year*
Media Lab Helsinki	2008
Degree Program	
Master of Arts in New Media, University of Art and Design Helsinki	
Author	
G. Martin R. Gimpl	
Title	
Stripes — A Conceptual Operating System User Interface	
Level	*Page count*
Master's Thesis	126

Abstract

Traditional computer operating system graphical user interfaces were designed decades ago have since become an integral part of our everyday lives. While fundamentally excellent, recent changes in the industry and use cases for computers are exposing the weaknesses in the implementations of current operating system user interfaces. This thesis is an attempt to propose an alternative, modernized user interface design — entitled *Stripes* — based on usability, simplicity, productivity, and other design guidelines.

The history and recent innovative additions to the current implementations of WIMP-based operating system user interfaces are outlined and analyzed. This contextual mapping of the field helps in identifying key problems and strengths of modern computer user interfaces, and, in addition to the aforementioned guidelines, form a basis for the design of the proposed concept.

The fundamental design choices, as well as several details of *Stripes* are described and explained. Its theoretical improvements and disadvantages over current user interface implementations are first discussed, then tested with the help of an interactive prototype. The favorable and critical test outcomes are described in detail, and reflected against the original design guidelines of the concept. Finally, conclusions on the success of the proposed interface are provided.

Keywords

User interface, operating system, concept, design

Osasto	*Vuosi*
Medialaboratio	2008
Koulutusohjelma Uuden median maisteriohjelma, Taideteollinen korkeakoulu	
Tekijä G. Martin R. Gimpl	
Työn nimi Juovat — konseptuaalinen käyttöjärjestelmän käyttöliittymä	
Työn laji Lopputyö	*Sivumäärä* 126

Tiivistelmä

Perinteiset käyttöjärjestelmien graafiset käyttöliittymät suunniteltiin vuosikymmeniä sitten ja ovat muodostuneet osaksi jokapäiväistä elämäämme. Viime vuosien tietoteknillinen kehitys ja tietokoneiden muuttuneet käyttötarpeet ovat tuoneet esille nykyisten käyttöliittymien heikkoudet. Tämä lopputyö esittää vaihtoehtoisen käyttöliittymätoteutuksen, nimeltään *Juovat*, joka painottaa käytettävyyttä, yksinkertaisuutta, tuotteliaisuutta ja muita käyttäjäkeskeisiä arvoja.

Työ kattaa ja analysoi nykyisten käyttöliittymien historian sekä viime vuosien tärkeimmät innovaatiot. Alan tutkiminen helpottaa nykyisten käyttöliittymien heikkouksien ja vahvuuksien kartoittamisessa sekä auttaa muodostamaan selkeät kriteerit suunnitellulle käyttöliittymätoteutukselle.

Juovat-käyttöliittymän kokonaisvaltainen suunnittelu ja useat yksityiskohdat esitellään seikkaperäisesti. Toimivuutta analysoidaan vertaamalla toteutuksen vahvuuksia ja heikkouksia olemassa oleviin käyttöliittymiin sekä testaamalla käytännön toimivuutta prototyypin avulla. Lopuksi testituloksia peilataan alkuperäisiin suunnittelukriteereihin ja pohditaan työn onnistumista.

Asiasanat

Käyttöjärjestelmä, käyttöliittymä, konsepti, suunnittelu

Foreword

For some unexplainable reason, operating systems and especially their user interfaces have been a focus on my interest for the past few years. Perhaps because, after a decade of stale development, a renaissance of some sorts has emerged — a newfound spark of interest amongst the public and developers to again invest their time and attention in how we interact and communicate with computers. Certainly, operating systems themselves are just a piece of the puzzle; but like any foundation, their importance is immense, both in terms of technical performance as well as how quickly and easily they allow us to accomplish the tasks we have set out to do. An operating system is at the very heart of any computer, and its user interface is our means to access all the power and functionality it has to offer. Thus, a poorly designed interface turns the computer, ultimately designed as an enabler, into a frustrating and inefficient hindrance; and as they are increasingly becoming an ubiquitous, integral part of our everyday lives, the need for simpler, more productive operating system interface design seems glaringly obvious. The concept presented in this paper, entitled *Stripes* for the time being, is an attempt to achieve this.

Needless to say, this project has been a true labor of love. I would like to thank the Media Lab at the University of Art and Design Helsinki for their support and giving me the opportunity to embark on such a personal, aspiring project. I would also like to thank my supervisors, Samu Mielonen, Markku Reunanen, and Rasmus Vuori for their dedication and feedback along the way — the end result would have been unspeakably worse without your input. Furthermore, I want to thank all of my prototype testers, Markus Dochnal, Max Dochnal, Jon Fabritius, Taskan Gokce, Anne-Marie Grönroos, Leena Honkanen, Jarno Koponen, Taro Morimoto, Antti Riikonen, Jaakko Riikonen, Stina Roos, Petri Rötkö, and Linda Söderholm for their time, feedback, and discussion. Also, I want to thank Eljas Perheentupa and everyone else at Linja Design, both for their flexibility and comments, Thomas Lambert and Richard von Kaufmann for proofreading this paper, as well as Ji Hyun Hong and Richard von Kaufmann for their role as opponents. Finally, thank you for everyone at Media Lab who patiently and politely listened to my three-hour presentations and still somehow found the energy to engage into discussion afterwards.

Table of Contents

1 Introduction

Over the last two decades, personal computing has become an integral part of our everyday lives. Computers are omnipresent, sitting on our office desks, residing in our living rooms, and traveling with us in our pockets. Computers have enabled technical innovation to change the way human beings interact with each other and the world surrounding them, becoming increasingly important tools for communication, commerce, consumption and the organization of entertainment or personal data, as well as networking.

However, the versatility and flexibility of the personal computer (PC) is also one of the main reasons for its complexity. Imagine a tool such as an axe with not just a single interaction method of swinging and cutting, but rather thousands of possible actions and uses, coupled with customizable preferences and an ever-increasing list of features, all encapsulated in a virtual environment that one cannot truly touch or feel. It is hardly a wonder why computers are often considered difficult to use despite all of the advancements in technology, but since they offer so much, humans have simply adapted to their ways, learned to use them, and accepted the status quo.

Most of the modern graphical user interface (GUI) implementations stem from the valuable research done in the 1960s and 1970s. Popularized in the last two decades, these aging designs, while functional, carry with them a heavy load of legacy and baggage, models and metaphors that made sense when computers had just been introduced to office workers. As development is racing on, however, early GUI conventions are becoming increasingly irrelevant and cumbersome while software is getting more full-featured and complex; an equation that inevitably leads — or, arguably, already has led — to overly complicated and unnecessarily frustrating experiences with operating personal computers.

The functionality of computers and the needs of users have changed considerably in the last few decades. By rethinking the traditional computer user interface (UI) models we have the possibility to reduce complexity and error rates, increase productivity and, most importantly, add to the pleasure of using personal computers.

This chapter presents an introduction to the project by defining its objectives, motive, design approach, target group, challenges, and scope. In addition, certain fundamental terms are defined in the context of this paper.

1.1 Definition of key terms in the context of this paper

This section provides definitions for often-used key terms in the context of this paper. Several broad terms such as "user interface" and "WIMP-based operating system" are also defined.

1.1.1 "User interface"

The term "user interface" can be applied both in terms of a physical (hardware) as well as a virtual (software) interface. Since this project focuses on the virtual part of interfaces, and since the proposed conceptual design should be usable on a variety of devices, future references to the term "interface" are limited to virtual interfaces exclusively.

Alan Kay, a pioneer of user interface design and object-oriented programming working for Xerox PARC and later Apple Computer, Inc. (among other companies and institutes), famously stated: "people who are really serious about software should make their own hardware [Folklore.org, 1982]." Coupling physical interface and interaction design with digital interface design allows for potentially superior solutions, since neither the physical nor the virtual interface has to be compromised to fit the restrictions of the other. In the real world, however, it is often difficult to introduce completely new physical interface designs that replace established standards — such as more productive keyboards — due to the learning curve required to use them after having being accustomed to prior solutions. A well-known example, the Dvorak keyboard, serves to prove the point; developed in the 1920s by one of the founders of industrial engineering, this improved keyboard layout never gained acceptance over the established QWERTY-layout, since its main benefit — moderately[1] increased typing speed — did not make up for the effort of completely relearning the use of a keyboard. [Norman, 1990, 147–150] [Shneiderman, 1992, 238–239]

1.1.2 "Usability", "utility", and "user experience"

Usability, utility, and user experience are terms that often intertwine with each other or get mixed up altogether. Their definitions in the context of this paper are described below:

- **Usability** and **utility**. Usability-engineering expert Jacob Nielsen summarizes the difference between usability and utility as follows: "utility is the question of whether the functionality of the system in principle can do what is needed, and usability is the

1 Norman quotes increased typing speeds of 10%, Shneiderman 33% (for expert users), in addition to reduced error rates.

question of how well users can use that functionality." [Nielsen, 1993, 25] In other words, utility describes *what* can be done, while usability defines *how* it is done. This is a necessary division, since the goal of this project is not to enhance utility, but rather usability — sometimes even at the cost of reducing the former in the process.

- **User experience**, on the other hand, is a wider, fuzzier term; it embraces both utility and usability, but also adds the overall experience or, for the lack of a better term, the general *feel* in the mix. User experience could be described as the sensation or impression the user has when using a system. Donald A. Norman, author of several award-winning books on usability and cognitive psychology, defines user experience as "[dealing] with all aspects of the user's interactions with the product: how it is perceived, learned, and used. It includes ease of use and, most important of all, the needs that the product fulfills [Norman, 1998, 47]." Another way to describe user experience is to liken it to user *satisfaction*, a subjective property that cannot be achieved solely by testing usability or providing utility. A positive user experience is a particularly desirable design goal that is very challenging to measure due to its emotional and psychological nature.

1.1.3 "WIMP"

The acronym "WIMP" stands for Windows-Icons-Menus-Pointer, and is often used to describe the current generation of computer user interfaces. Before the WIMP paradigm shift took place, as described in more detail in section 2.1, early computing systems were operated via punch cards, often with the help of a dedicated operator person, or via a more direct command line interface (CLI). The original WIMP model has changed surprisingly little since it's popularization in the mid-1980s. [van Dam, 1997, 63]

In the context of this paper, WIMP is used as an umbrella term for the traditional metaphors and user interface elements that are widely used in modern operating systems GUIs, including the Desktop. Terms such as "WIMP-based operating system" refer to the *implementation* of the WIMP metaphor, not to the acronym itself.

1.2 Design objectives

This section presents an overview of the goal, focus areas, and the vision of the project. The final project deliverables are also outlined.

1.2.1 Goal

The goal of this project is to critically judge the norms, models, and metaphors used in the implementations of current computer graphical user interfaces[2] and propose improvements and alternative implementations in the form of a conceptual operating system user interface design, complete an interactive prototype, and present preliminary user test results based on that prototype.

On a personal level, the goal of the project is to become a respectable reference work that could account for further academic and/or commercial pursuits in the field of interface or interaction design, demonstrating both theoretical as well as practical understanding of user interface design.

1.2.2 Focus areas

There are several focus areas to be addressed that outline the goals of the proposed interface design. These focus areas form the scope of the project, and provide a framework of guidelines for any specific design decisions:

1. First and foremost, the **reduction of complexity and management** present in current systems, even with the cost of lessened functionality, is a top priority. Minimizing the amount of user interface elements and operations helps decreasing cognitive load, and allows for focusing on the task at hand. Increased simplicity also has the added benefit of reducing error rates.

2. The interface should promote **increased productivity and efficiency** when using the computer, enabling fast means to navigate, access information, and issue commands.

3. Highly **usable and satisfying**, the interaction between the system and the user should be a seamless, non-obtrusive, and pleasant affair. A computer might essen-

2 The term “Computer graphical user interfaces”, in this case, refers to so-called desktop operation system GUIs, used in traditional desktop computers, notebooks, and Ultra-Mobile PCs (UMPCs), *not* devices such as mobile phones or mobile multimedia players.

tially be a tool, but it can also be regarded as a partner — a responsive system that is there to help to achieve the tasks of the user.

4. Instead of providing layers upon layers of customizability, an adaptive interface focuses on **learning from its user**, and is dynamically altered based on the user's habits in order to emphasize popular actions, further streamlining and simplifying interaction with the system.

5. A formidable interface should be flexible enough to be both **accessible for novice users and powerful for expert users** at the same time, as well as aid users in becoming more productive as their skills increase — essentially promoting the most efficient use of the system.

6. A certain level of audio-visual **attractiveness** is required for producing an interface solution that grabs people's attention and provides enough incentive for them to even consider going through the trouble of using it. Aesthetic design also increases the overall user satisfaction of a system.

7. Due to the ever-expanding range of personal computers, an interface should be able to **accommodate** itself to reasonably small[3], handheld screens and multi-screen setups, and be controllable via touch, mouse, and keyboard.

8. Instead of a radical design that is only feasible after several years of technological innovation, the proposed interface solutions should be **achievable with today's technology.**

This compiled list of focus areas is similar to Nielsen's definition of usability, which consists of the following five attributes: learnability, efficiency, memorability, errors, and satisfaction [Nielsen, 1993, 26]. While Nielsen opts for a set of measurable aspects of usability, criticizing the pursuit of "warm, fuzzy [...] 'user friendliness'" [Nielsen, 1993, 27], the focus areas for this project indeed include *both* general usability and attractiveness. While many usability experts and graphic designers often pay attention to solely their respective fields, striking a balance between the two, as well as other disciplines such as social and behavioral sciences, is what ultimately defines the user experience of a system [Norman, 1998, 48–49]. Humans are attracted to aesthetics by nature, but beauty is only skin-deep unless the underlying system provides usability and utility to actually perform the given

3 "Small" is, of course, a very relative term; the minimum target resolution is 800 pixels in width by 480 pixels in height, a typical UMPC screen resolution. Smaller screen resolutions a better served with a dedicated, device-specific user interface that is specially tailored to meet the device's needs and restrictions.

tasks in a pleasant manner, even after months of use. Norman points out that attractive products actually work better than unattractive ones. He argues that when users feel good using a product, they overlook design faults and find the general behavior of the system to be effortless. [Norman, 2002, 41] The biggest challenge is to find the optimal balance between aesthetics and usability, where neither one is drastically compromised because of the other.

Another important aspect of a successful interface is the conceptual model that it provides, and how well this model is transmitted to end-users. Conceptual models could be described as individual interpretation and the overall understanding of the system in use, and end-users generally form their conceptual model based on their prior experience with similar systems in the past, training, and instruction [van der Veer, et al., 2002, 52–80] [Norman, 1990, 16–17]. One of the priorities in designing a usable system is to successfully establish the correct model in the users' minds, since failing to do so hinders learning and causes unnecessary confusion, frustration, or doubt in the established conceptual model [Shayo & Olfman, 1998] [Norman, 1990, 189–190]. Since most users are already familiar with certain norms of computer user interfaces, and have formed habits based on their functionalities, the proposed design should convey an immediately understandable conceptual model, even if it differs from prior implementations.

1.2.3 Vision

The key focus areas described in the previous subsection could also be considered as the collective "vision" of the project; however, they form merely an unproven basis for the desired interface design. User tests, and the actual reception by users and the consumer market, will ultimately dictate whether or not the proposed vision is actually considered viable or useful.

Nevertheless, a proper vision is crucial to the success of any design process. In the field of user interface design, one of the best examples of the importance of vision-driven design comes from the open source project Firefox. Blake Ross, co-founder of the software, explains in his online article *The Firefox Religion*:

> "Firefox has humble origins in a product that — if everything went as planned — was designed to be *invisible* to the person using it. I remember sitting on IRC with [co-workers] Dave, Ben and Asa painstakingly debating feature after feature, button after button, pixel after pixel, always trying to answer the same basic question: does this help mom use the web? If the answer was no, the next question was: does this help mom's teenage son use the web? If the answer

was *still* no, the feature was either excised entirely or (occasionally) relegated to config file access only." [Ross, 2005]

The Mozilla web browser, a competing[4] software suite to Firefox, on the other hand, lacked a clear vision. Ben Goodger, another co-founder, expounds in his online article *Where Did Firefox Come From?* on the *Inside Firefox* blog that there was no organized vision for the user interface of Mozilla. The ability to make changes was, apparently, so widely distributed that anybody could make an addition to the UI. The result was a cluttered interface that lacked vision. [Goodger, 2006]

Although there are several reasons why Firefox is currently considered the more popular product of the two, with nearly 30 times the amount of users according to browser statistics provided by W3Schools [W3Schools, September 2007], there is no denying that having a clear vision has helped define and simplify the Firefox browser; as Goodger explains in another *Inside Firefox* blog post, *Why the Mozilla Browser Sucks*, "what is necessary is either a small group of developers with a well understood and focused vision, or a set of UI specs up front [Goodger, 2003]." While having a clear vision does not guarantee success alone, it aids in the design process by justifying design decisions and keeping the product true to itself (i.e. its goals) and, by extension, reduces the risk of unnecessary complexity, feature-creep, and bloat. An established vision also helps in parsing through user feedback and deciding which comments are the most relevant for the given project.

The vision of this project could be summarized into following sentence: "provide a user interface that improves upon prior implementations, if possible, in the context of the established focus areas." Accomplishing this goal might require breaking certain GUI standards, but such measures should always be justified by increased usability or user experience. To quote user interface design pioneer Jef Raskin: "[where] real improvement can be achieved by making major changes, the interface designer must balance the legitimate use of familiar paradigms, which ease the learning process, against the enhanced usability that can be attained by abandoning them [Raskin, 2000, 4]."

1.3 Target group

Since computers are used by virtually everybody in technologically advanced nations, the target group for the proposed user interface design is extremely wide by nature. As specified in the focus areas in subsection 1.2.2, the system should appeal to novices and experts

4 It could be argued that since both the Mozilla and the Firefox browsers are essentially based on the same rendering engine and base they are in fact not competitors, but rather different incarnations of the same product.

alike; therefore, it is not possible to define specific target groups as far as general computer skills are concerned. The main target audience, nevertheless, are consumers — people who have lightweight computing needs and possibly only a limited interest in the technical side of computers, thus most likely to benefit from a simpler, more usable user interface paradigm.

While computer experts, people with disabilities, the elderly, et cetera are not *specifically* targeted, that is not to say they are excluded either, since many design decisions and focus areas of the proposed user interface aim to improve its implementation at a very human, cognitive level, and therefore can hopefully be universally appreciated. Naturally, culturally dependant design changes will have to take place if the system aims to be internationally usable. Although it is naïve to expect everyone to relish the proposed design, nobody should feel excluded per se.

1.4 Motive

One of the most profound questions surrounding this project is not *how* to design a new interface paradigm, but rather *why* it should be done. Do we really need something different than what is being offered, are there any real benefits to be gained, and even if there are, are they really worth the effort? After all, the traditional WIMP interface in combination with its desktop metaphor, now taken for granted, has served us well for over two decades.

The landscape of personal computing has, however, started to change radically. The Internet, for one, has completely altered the way many people use the computer — and for what purpose; it has expanded the means of communication and acquiring information, is increasingly being used for consuming media, managing personal data, and creating social networks, and is transforming into the center of our digital lifestyles. The current advancements in mobile phone, game console, and portable media player industries are bridging the gap between consumer electronics and personal computers. While many of these miniature computers do not directly compete with the personal computer, they are providing alternative interfaces, albeit often heavily inspired by the WIMP model, that arguably provide a much simpler, easier user experience than the traditional operating systems on personal computers. The current operating system user interface implementations, burdened by their established heritage of aging metaphors and functional models, face challenges in providing a truly simple, usable computing environment. A fresh approach is needed if such a goal is of any value. To paraphrase Nielsen, due to the increasing computing power, decreasing hardware prices, and ubiquity of computers, it now pays to dedicate computational resources to making life easier for the users [Nielsen, 1993, 8].

Usability has long been considered as a secondary objective in most software (and even hardware) designs due to the feature-driven nature of the industry. Getting a product to function properly in technical terms is, understandably, always the first priority. This is a simple rule of logic; utility consistently overshadows usability since usability in itself doesn't get anything *done*. This is the great paradox of user-centered design; everyone agrees on its importance, but ultimately it is not what sells the product to consumers. It remains to be seen whether or not this utility-driven development persists in the future, but as software titles progressively grow more and more complex due to their ever-increasing list of features, there is reason to believe otherwise. Norman points out the changing characteristics of the wants and needs of users during the course of all technological progressions: at the young stages of development, early adopters with great interest in new technology drive the demand for features and other technological advancements, but as soon as technology matures, a far larger segment of the market, the late adopters, start taking the technology for granted and insisting on convenience and user experience instead [Norman, 1998, Preface ix–x].

Furthermore, Norman wonderfully exposes the paradox of computers. On one hand, they are an area where all the major difficulties of design can be found; design of the systems is left to engineers and programmers, experts in the field of computing, and not to usability professionals who understand the requirements of human-centered design [Norman, 1990, 177]. On the other hand, however, there is much potential in computers: due to their unparalleled power, adaptability, and utility they are able to make life easier for humans [Norman, 1990, 179]. All that needs to be done, really, is to shift focus from utility to usability.

Colin Ware, doctor of psychology and expert in data visualization, underlines the overall importance of advancing digital user interfaces from the point of view of the human cognitive system:

> "[Efforts] to refine the user interface are extremely important. One of the goals of cognitive systems design is to tighten the loop between human and computer, making it easier for the human to obtain important information from the computer via the display. Simply shortening the amount of time it takes to select some piece of information may seem like a small thing, but information in human visual and verbal working memories is very limited; even if a few seconds of delay or an increase in the cognitive load, due to the difficulty of the interface, can drastically reduce the rate of information uptake by the user. When a user must stop thinking about the task at hand and switch attention to the computer interface itself, the effect can be devastating to the thought process. The result can be the loss of all or most of the cognitive context that has been set up to solve the real task. After such an interruption, the train

> of thought must be reconstructed, and research on the effect interruption tells us that this can drastically reduce cognitive productivity." [Ware, 2004, 350]

As the personal computer edges closer and closer into our pockets, further away from that lonely, murky downstairs corner, it becomes increasingly important to consider alternative design approaches that can accommodate to the broadening world of personal computing without being so complicated to use.

1.5 Design approach

With any user-centered design approach, it is only logical to expect the design process to be fueled by inquiring users for their requirements. After all, if end-users are indeed the main focus of the proposed design solution, why not simply ask them what it is that they want?

Unfortunately, as Nielsen states, users do not often know what is good for them:

> "One example is a study of the weight of telephone handsets conducted in the late 1950s when people were used to fairly heavy handsets. The result of asking users whether they would like lighter handsets was no, they were happy with [what] they had. Even so, a test of handsets that looked identical but had different weights showed that people referred handsets with about half the then-normal weight." [Nielsen, 1993, 11]

The issue boils down to the lack of references. As long as there are no better solutions on the market, people have a hard time imagining them and, by extension, demanding them.

Another issue is subjectivity: there are different needs and different priorities for every single user. It is simply impossible to design a system that accommodates each and everyone perfectly, but individual users often only have a narrow point of view — their own. It takes objectivity and certain professionalism to be able to stand back and see the big picture instead of personal preference.[5] Users are not designers, therefore their role in providing feedback rather than direct design guidance is far more beneficial.

Due to the aforementioned reasons, the approach to this project is not based on preliminary user tests or questionnaires, but rather on common usability guidelines and the aforementioned vision. That is not to say that tests are not an important factor in *evaluating* a design;

5 Personal preference can never be completely avoided; even the most skilled designers have their individual bias. Still, designers possess extensive knowledge and, more importantly, perspective on different design solutions and their respective pros and cons.

end-users have every right to complain about poor design, but they should not be the ones to design the solutions to those problems — that is the task of professionals, and applies to the field of user interface design as much as to any other.

1.6 Challenges and scope

There are several challenges associated with this project that limit its goals, scope, and results. While significant, they should not invalidate the project.

1.6.1 Challenges

One of the most challenging aspects of designing a new user interface is to hit the right balance between introducing novel, useful concepts and providing enough familiarity. As most humans tend to resist change, especially if it means unlearning old knowledge, the proposed design must show immediate usefulness and attractiveness to novice and expert users alike. It must be different enough to cause genuine interest upon first sight, but avoid being so radical that it becomes perplexing or unbearable to understand.

A prominent challenge, as a designer, is to avoid the common pitfall of becoming too familiar with one's own design and still considering oneself as a typical user. As Norman points out, designers "become so proficient with the product [they designed] that they can no longer perceive or understand the areas that are apt to cause difficulties [Norman, 1990, 156]." It is essentially impossible to design a product and *not* be thoroughly familiar with it, and this extensive knowledge is exactly what many of the product's end-users are lacking. This is also why user-testing plays such an important role in product evaluation, since it is one of the most efficient ways to gain valuable feedback from people who have never used the product before.

There are, of course, a multitude of technical challenges if the proposed system was ever to be created as an operational, working product. In addition to the enormous task of coding the operating system, the development of third-party applications is a major concern; few platforms have survived long without the added value of software applications written specifically for them. Although it would theoretically be possible to provide simple solutions for most common tasks, such as web browsing and email, a major shift in interface design would require not only the acceptance of end-users, but also capture the hearts of developers. In fact, getting developers interested in a platform is a necessity for gaining any interest among the general public. In an ideal situation, existing software could simply be converted to fit in the new environment with the help of development tools, but this is hardly

a realistic presumption. Fortunately, in the age of the platform-independent Internet, web-based applications can counter the lack of some platform-specific applications.

Finally, it could be argued that the very notion of creating another operating system user interface design is a failed attempt from the get-go. Norman, in his book *The Invisible Computer*, argues very forcefully that the problems of the personal computer are of fundamental nature, and not solvable without the death of the PC as we know it: "the problems [of the PC] result from the very power of a general purpose machine and the attempt to make this single device, with a simple keyboard, display and pointing device, serve all needs [Norman, 1998, 111]."

1.6.2 Scope

By no means will it be possible to produce a commercially available product or even a release candidate. An operating system for a personal computer is, after all, a vast[6] and highly complicated software system that manages countless tasks and operations. The prototype will not be fully interactive, but rather provides just enough functionality for any meaningful testing. Furthermore, many of the concepts presented are influenced by prior interface designs; existing solutions and concepts, outlined in sections 2.3 and 2.4, will be recognized and credited when possible.

A truly groundbreaking interface design that challenges the WIMP metaphor itself, not its current implementations, is hardly possible without moving beyond the associated keyboard/mouse combination of input devices, or the 2-dimensional displays in use today. As such, this project aims to provide an achievable upgrade to the current model rather than propose a truly radical redesign requiring the use of cutting-edge technology and major innovation in physical interface design.

6 "Vast", as in several *million* lines of code. According to Vincent Maraia, author of the book *The Build Master: Microsoft's Software Configuration Management Best Practices (The Addison-Wesley Microsoft Technology Series)*, Microsoft Windows Server 2003 contained no less than 50 million lines of code. [Knowing.NET, 2005]

2 Context

This chapter provides the contextual background for the project, summarizing a brief history of operating system user interfaces, identifying the problems of their current implementations, and presenting alternative approaches and innovative additions to the WIMP model. This research forms a basis for the proposed user interface concept outlined in chapter 3.

2.1 Brief history of operating system user interfaces

The graphical user interface has quickly become the standard interface concept for controlling a personal computer, and thus is often taken for granted, despite the fact that the GUI was commercially introduced less than 30 years ago and popularized ten years later. This standardization has become so encompassing that it is easy to forget how much of their success modern GUI implementations owe to the strong underlying design work and conceptual models pioneered in the very early days of computing. Therefore, a concise summary of the most important UI concepts and general GUI development is helpful in order to understand the current Context and status quo of GUIs on personal computers.

The development of user interfaces on computers can roughly be divided into three distinct phases: the batch, the command line interface (CLI), and the graphical user interface (GUI) eras [Raymond & Landley, 2004]. Of course, it is a gross simplification to make such strict distinctions, especially when none of these interface approaches can be considered obsolete or irrelevant even in today's computing landscape, but they aid in laying out a coherent overview of the different paradigms of user interface design.

2.1.1 Batch computing and command line interfaces

Early computing systems, used during the Second World War in the mid-1940s, were called batch computers; they were operated with stacks of custom-manufactured punch cards fed into the machine in one go, hence the name. Due to the limited hardware and processing power, batch computers did not work in real-time, and were often used by dedicated system operators. In the late 1950s, special monitoring programs were added to batch computers that resided on the machine itself, enabling easier maintenance, error checking, feedback, and support for services and I/O operations — fundamentals of operating system functionality. [Raymond & Landley, 2004] [Stephenson & Birkel, 2004]

When computers were connected to teletypes in the 1960s and 1970s, a more direct and efficient way to operate computers emerged, marking the dawn of a new era in computing paradigms: command line interfaces. The interaction model of CLIs was a series of typed commands, each operation executed after another using specialized vocabulary. Once

video-display terminals became more common, the need for supplementary material (first cards, then paper and ink) disappeared. The highly influential CLI-based operating system, Unix, was created in 1969, and Digital Research's CP/M (Control Program/Monitor) and Microsoft's MS-DOS (Disk Operating System)[7] followed in the next decade. [Raymond & Landley, 2004] [Stephenson & Birkel, 2004] [Carter, 2005] Although the popularity of command lines was soon to be surpassed by the next paradigm shift in user interfaces, the direct, efficient, and exposed nature of CLIs remained attractive to many enthusiast and expert users; even today, most modern operating systems contain a fully-functional CLI terminal.

2.1.2 Origins of graphical user interfaces

The third era, the age of the graphical user interface, was popularized in the mid-1980s, but has much deeper roots in history. Some of the fundamental elements of GUIs were already envisioned by Vannevar Bush in the early 1930s — long before computers even existed. Bush described a device called the "Memex", an "enlarged intimate supplement to [an individual's] memory [Bush, 1945, 6]" as he described it in his article *As We May Think*, published in the *Atlantic Monthly* in July 1945. The Memex, as illustrated in Figure 1, was essentially a large desk-like storage cabinet for media, operated via a keyboard, levers, and buttons, and provided "translucent screens, on which material can be projected for convenient reading [Bush, 1945]." [Reimer, 2005]

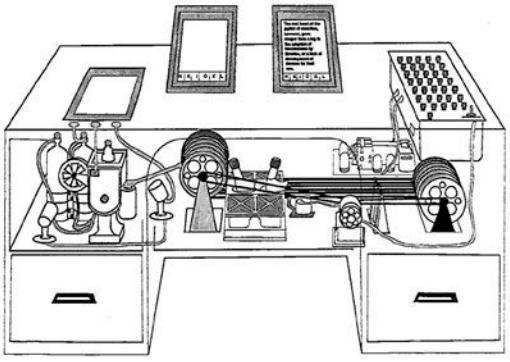


Figure 1 **The Memex as described by Vannebar Bush.**

The visions of Bush were picked up by Douglas Engelbart, who published his groundbreaking article *Augmenting Human Intellect* in October 1962 on "improving the intellectual effectiveness of the individual human being [with the help of a] computer, [that] can be harnessed for direct on-line assistance, integrated with new concepts and methods [Engelbart, 1962]." Six years later, in 1968, Engelbart and his team from Stanford Research Institute demonstrated their ideas publicly at Menlo Park, California by introducing the oNLine System (NLS). Along with other inventions such as the mouse and a 5-key chord keyboard, Engelbart introduced the first prototype of a graphical user interface, as presented

7 DOS is a family of early CLI operating systems. A version of DOS, MS-DOS, was used by Microsoft and later became the basis for several incarnations of the Windows operating system.

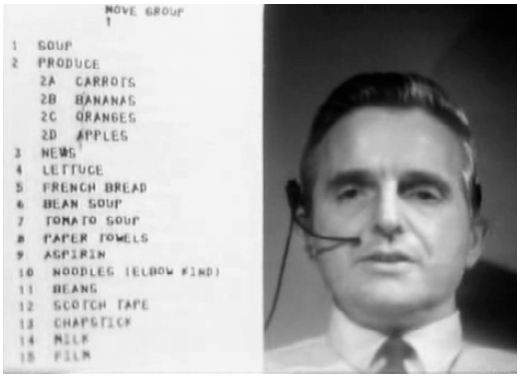


Figure 2 Douglas Engelbart demonstrating the NLS in 1968.

in Figure 2. NLS included a mouse pointer, rudimentary windows, and menus in addition to incredibly advanced features for the time such as full-screen document editing, hyperlinks, real-time collaboration, electronic mail, instant messaging, and even video conferencing. [Reimer, 2005] As Engelbart was working on NLS, another pioneer, Ivan Sutherland, presented his Sketchpad system — a man-machine graphical communication system — built for developing military radars. Sketchpad introduced a host of well-known tools for drawing on a computer, such as rubber-band lines and constrained drawing[8], among others. In addition, Sketchpad treated its graphical elements as objects, laying the foundation for object-oriented programming languages. [Wardrip-Fruin & Montfort, 2003, 109]

2.1.3 GUI research and its emergence in operating systems

Engelbart's NLS project eventually ran out of funding, but his demonstration had made a large impact on Xerox, a company mainly known for its paper-based photocopiers. The company started a research division called PARC (Palo Alto Reseach Center) in 1970, which attracted many skilled and visionary researchers, including several of Engelbart's employees. In 1973, PARC introduced the Alto computer, designed as a complimentary system to drive the newly invented Xerox laser printers (as evidenced by its distinct vertical display). The Alto was able to present bitmapped raster images (as opposed to vector-based systems such as the NLS), and included a mouse and keyboard. What Alto lacked, however, was a consistent GUI for software applications — a problem that was remedied a year later by the introduction of Smalltalk, a so-called graphical development environment for programmers. [Reimer, 2005] One researcher in particular, Alan Kay, contributed to the emerging GUI metaphor, shown in Figure 3, by implementing overlapping windows and the desktop metaphor [Johnson, 1997,46–48]. In addition, Smalltalk introduced the concept of icons along with the possibility to directly manipulate them, as well as contextual menus, scrollbars, dialog boxes, and radio buttons — essentially the backbone of what is considered a WIMP-based graphical user interface. [Reimer, 2005] Smalltalk, partly inspired by

8 Rubber-band lines refer to lines that change dynamically even before the end-point of the line has been defined. Constrained drawing refers to the ability of the system to create clean shapes even when the input is not exact, such as forming perfect circles with less-than-perfect input.

Sutherlands Sketchpad, is also considered one of the first object-oriented programming languages [Wardrip-Fruin & Montfort, 2003, 109]. Kay and fellow PARC researcher Adele Goldberg further promoted the idea of computers as personal tools and envisioned notebook computers; in their article *Personal Dynamic Media* published in March 1977 in *Computer*, Kay and Goldberg describe the Dynabook, a mobile, general-purpose device for creative use — a radical paradigm shift in what was considered to be the use of computers at the time. [Wardrip-Fruin & Montfort, 2003, 391].

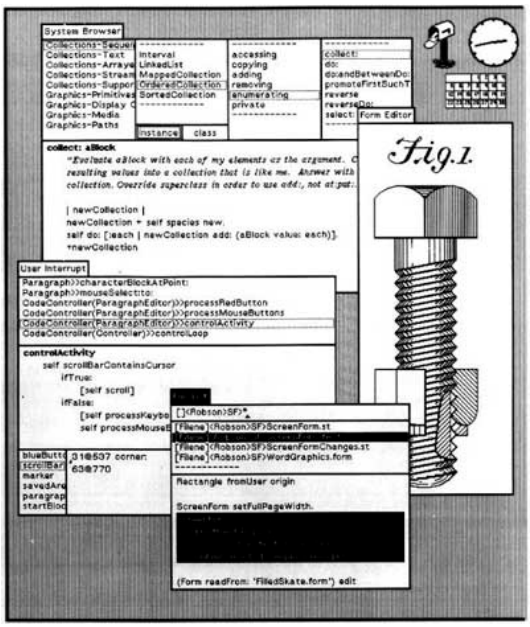


Figure 3 The Smalltalk GUI.

In the early 1980s, when GUI research was at its most driven and inspired, the notion of direct manipulation, first introduced by Sutherland's Sketchpad, re-emerged. Direct manipulation — as described by Ben Shneiderman in his article *A Step Beyond Programming Languages*, published in August 1983 in *IEEE Computer* — was the opposite of CLI-based interaction: instead of employing a (textual) command language, the computer was controlled "directly"[9] , as shown in Figure 4, by pointing at graphical objects and selecting commands. [Wardrip-Fruin & Montfort, 2003, 485]. Direct manipulation was one of the major shifts in user interface philosophy and would soon spur the emergence of personal home computing. Unfortunately for Xerox, the Alto was never commercialized, and its descendant the Star failed in the marketplace; however, their GUI inspired Apple Computer's co-founder Steven Jobs during a tour around the Xerox PARC facilities. Funded by their success of the Apple II computer, and reinforced by several ex-PARC employees

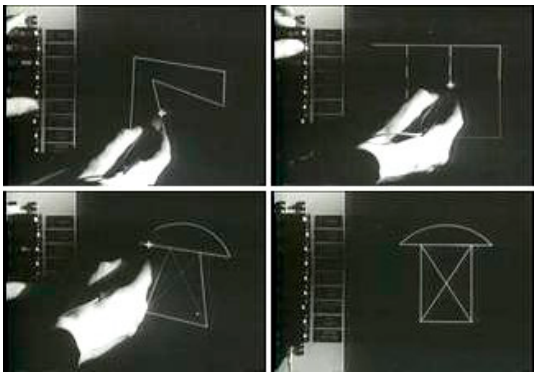


Figure 4 Direct manipulation in Ivan Sutherland's Sketchpad.

9 Ironically, from the point of view of a computer system, command line interfaces are in fact far more direct than graphical user interfaces. Psychologically for the user, however, GUIs *feel* more direct, since objects are manipulated by pointing-and-clicking instead of typing in paths and commands.

including Kay, Apple's team started developing their own GUI-based operating system that would eventually become the Macintosh System 1 operating system[10]. The Mac OS, introduced with the original Macintosh personal computer in 1984, included such concepts as a static, consistent menu bar with drop-down menus along with keyboard shortcuts, the trash can, double-clicking on documents to open them, more advanced drag-and-drop file manipulation, and was operated with the polarizing one-button mouse. [Reimer, 2005] The Mac OS, as presented in Figure 5, was the first truly successful GUI-based operating system and can be considered to have set the standard for other GUIs to come [Johnson, 1997, 49].

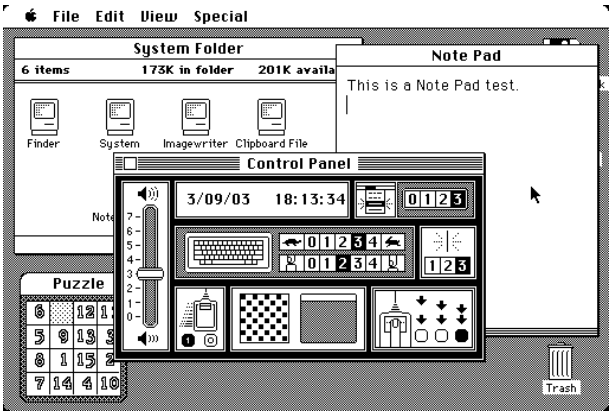


Figure 5 The Macintosh System 1 operating system GUI.

2.1.4 Popularization of GUIs

Apple was certainly not the only company working on GUIs in the 1980s; as the GUI approach started to gain in popularity, many players in the industry wanted a piece of the market — unfortunately without driving much innovation in the process. During the first half of the decade, VisiCorp's VisiOn, and Digital Research's GEM (Graphical Environment Manager)[11] appeared for the IBM PC platform that had thus far been powered with CLI-based operating systems such as CP/M and MS-DOS. Several GUIs were created as graphical shells running on top of DOS, such as the original Microsoft Windows 1, and Tandy Computer's DeskMate. Berkeley Software provided a graphical shell, the GEOS (Graphic Environment Operating System), for the widely successful Commodore 64 computer[12], and Commodore's Amiga computer included the Amiga Workbench graphical operating system, as displayed in Figure 6. [Reimer, 2005] [Carter, 2005] [Lineback, 2007] While many of these "alternative" GUIs to the Macintosh OS were not quite as sophisticated, the

10 Prior to Macintosh System 1, Apple was working on the Lisa operating system, bundled with their Lisa computer. Lisa failed in the market, but its GUI lived on in the form of System 1.

11 The GEM operating system was also used on the Atari ST, released in 1985 by Atari Corporation.

12 GEOS was also ported to run on the Apple II, and later IBM-compatible PCs.

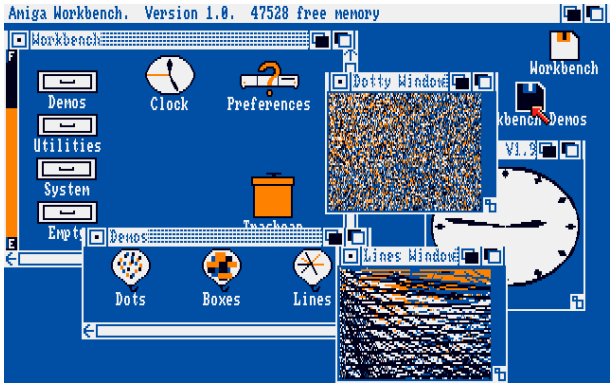


Figure 6 **The Amiga Workbench GUI.**

low price of hardware and compatibility factors helped them gain traction over Apple's relatively closed ecosystem. At the same time, the free[13] software philosophy was set in motion with Richard Stallman's *The GNU Manifesto* in 1985. Stallman, a member of a group of MIT hackers disgruntled by the increasing amount of closed-source software at their lab, decided to offer a compatible, free alternative to the commercialized UNIX platform. [Wardrip-Fruin & Montfort, 2003, 543]

During the late 1980s, GUI-based operating systems continued to refine and take over the overall computing market. Acorn Computer released the Arthur operating system along with their RISC-based computers, which later became a part of RISC OS, as presented in Figure 7. Arthur introduced the dock concept, a placeholder and launcher for commonly used programs, as well as anti-aliased font rendering. IBM's and Microsoft's joined GUI effort, OS/2, was released late in the decade, and was similar to Microsoft's evolving Windows platform. After resigning from Apple in 1985, Jobs established another company, NeXT Computer, and its promising, UNIX-based NeXTSTEP operating system. Late in the decade, other Unix-based computers started adopting GUIs as well; unlike other systems, however, many UNIX-based GUIs were created on top of common, networked windowing architecture, the X Windowing

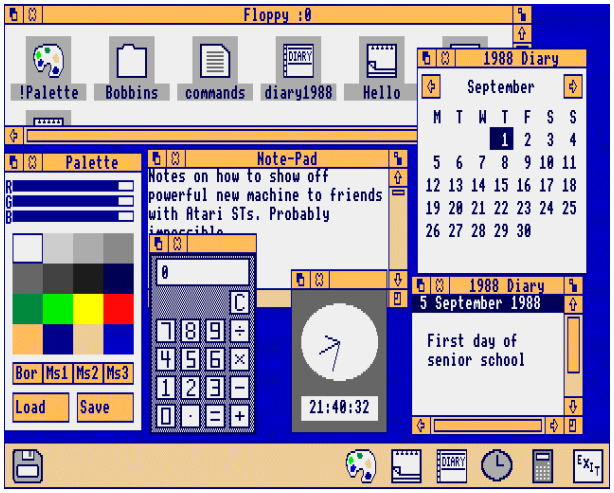


Figure 7 **Arthur's successor, the RISC OS GUI.**

13 The term "free" is often misunderstood in the context of free software, referring more to *freedom* than price. According to the manifesto, programmers should and need to be paid for their work in order for development to continue. Of course, programmers also have the right to distribute their code *free of price* if they so choose.

System[14] — enabling developers to build their own GUIs on top of the standardized framework. Two projects in particular, KDE and GNOME, were established as popular desktop environments; others included Sun Microsystem's Solaris, and Open Source Foundation's Motif. [Reimer, 2005] [Carter, 2005]

2.1.5 Standardization of GUIs

By the mid-1990s, several of the companies creating GUIs had vanished due to the rising popularity of Microsoft Windows. Early in the decade Microsoft released Windows 3.0 and 3.1, and the launch of Windows 95 skyrocketed the product's popularity — despite the fact that is was still heavily based on MS-DOS. Windows 95 included many new user interface concepts, such as the Start-menu for launching applications and the Taskbar for switching between them. At the same time, Microsoft was already developing a completely new Windows platform, the NT (New Technology) operating system. They also experimented with alternative GUIs, such as Microsoft Bob, a simplistic graphical shell running on top of Windows that replaced the traditional WIMP-interface with rooms and helper agents. Later in the decade, Be Inc. introduced BeOS — as shown in Figure 8 — an advanced operating system developed first for Be's own BeBox computer running the PowerPC architecture, and later for Intel x86. [Reimer, 2005] [Lineback, 2007]

The evolution of GUI concepts slowed down considerably after the mid-90s, partly because of the stagnating OS market due to an established and potent market leader, but the advent of the Internet and Internet browsers later in the decade created a whole new playing field for GUI experimentation and development. Also, the need for mobile GUIs running on limited hardware configurations started emerging, partly contributing to the rising popularity

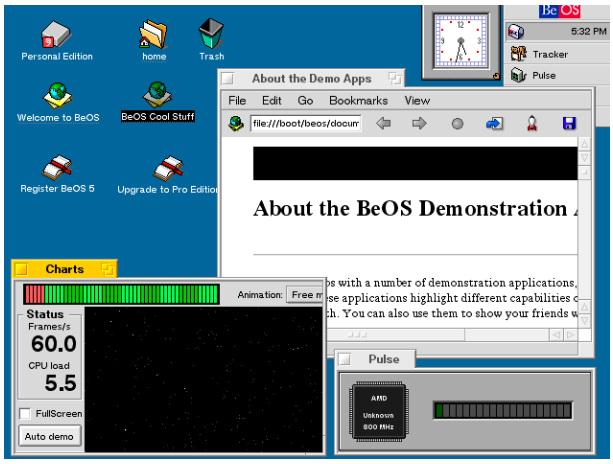


Figure 8 Be Inc.'s BeOS GUI.

14 The X Window System, also referred to as X11 or simply X, is a system for managing graphical user interfaces. It is responsible, among other things, for tracking applications and all their respective windows, as well as drawing them. [The Linux Information Project, 2006]

of the free UNIX-like operating system, GNU/Linux[15]. [Carter, 2005] Linux — similar to its commercial brethren, Unix — can run several GUIs (or simply its CLI), and comes in various distributions.

Early in the new millennium, Microsoft released Windows 2000, an NT-based operating system, as well its successor, Windows XP, that introduced the new platform to the general public and continued Microsoft's dominance in the marketplace. Apple Computers' Mac OS had been evolving steadily since its introduction in the mid-80s, but the company was looking for a successor for its aging platform. Instead of purchasing BeOS, Apple merged with NeXT, and used NeXTSTEP as the foundation for its Mac OS X operating system, as presented in Figure 9, released in 2001. [Reimer, 2005] Over the next years, Mac OS X introduced several new and revamped GUI concepts such as the Dock, Sheets, Exposé, Virtual Desktops, Time Machine, and others. Later in the decade, Apple introduced an innovative touch-based mobile platform built on a lightweight version of Mac OS X.

Figure 9 The Mac OS X Public Beta GUI.

15 Linux is, in addition to the kernel, contains several applications and commands originally developed for the GNU operating system created by the Free Software Foundation. Thus, it is sometimes referred to as GNU/Linux. [Weeks, 2004]

Since the original introduction of the GUI, there has been a steady evolution to improve and extend the WIMP model, but no radical 4th era of OS user interfaces. Experimental Virtual Reality (VR) systems point to immersive 3D environments, and several attempts have been made to eliminate the need of conventional file systems. [Raymond & Landley, 2004] Touch and other sensor-based interfaces propose novel and exciting new ways to interact with computers, while the rise of Internet-based applications foreshadow a resurgence of thin clients and terminals. It is clear that as computers become increasingly integrated with all aspects of our daily lives, their diversity calls for both new approaches as well as improvements to old models alike.

2.2 Problems of current WIMP-based OS user interfaces

Clearly, if history is of any indication, both the underlying WIMP model and its current implementations have certainly proven their strength and flexibility in the broad field of computing. However, as always, there is room for improvement and development. The dominant so-called desktop operating system platforms[16] currently on the market — Microsoft Windows, Mac OS X, and the various desktop Linux distributions — are all heavily based on the traditional, long-standing implementation of the WIMP model. While each platform has, over the years, refined and developed its unique features and user interface approaches, they all inherit the same problems (as well as benefits) stemming from their common legacy.

Outlined here are the most significant setbacks evident in current WIMP-based OS user interfaces, along with their implications. They form a basis for the design of the proposed, conceptual OS user interface described in detail in chapter 3, where both its pros and cons are evaluated against the current implementations. The list is presented in an alphabetical order.

2.2.1 Desktop metaphor

The desktop metaphor has existed in WIMP-based user interfaces since the dawn of the GUI; it was first introduced as a part of the Smalltalk operating system in the late 1970s by Alan Kay and his team working at the Xerox Palo Alto Research Centre (Xerox PARC) [Preece et al., 1994, 17]. Since computers were not common at that time, and the more intuitive GUI paradigm — as opposed to CLI — was only being developed, it was logical

16 The term “desktop operating system” is somewhat misleading, since they are not exclusively used on desktop systems. The term is used here in order to exclude mobile phones and other small-screen mobile devices.

to simulate the real world office, and thus built on users' existing mental models: enable interaction that was as similar as possible to what office workers were used to [Johnson, 1997, 48]. These electronic counterparts to the physical objects in an office included files inside folders, trash cans, and, of course, the actual workspace — the top of the desk [Preece et al., 1994, 145].

Today, 30-odd years after the invention of the desktop metaphor, we find ourselves in a very different computing environment. Computers are not anymore sold solely for work-related use that requires a work desk, nor do they revolve around files and folders: instead, they are increasingly used for consumption of media and accessing the Internet. The Desktop is hardly considered as a workspace anymore, instead it holds a background image and, often, a myriad of shortcuts or orphaned files for quick-launching purposes. The Desktop is so often cluttered that Microsoft created a special helper Wizard[17] for cleaning up unused Desktop items. In their recent revision of Mac OS X, Apple decided to place downloaded files in their own special folder instead of the Desktop, mainly due to their invasion of Desktop space. One of the main reasons for ending up with a cluttered Desktop, such the one illustrated in Figure 10, is simply the tediousness of file management: keeping the Desktop clean requires manual labor on the user's part, even if — or especially when — it involves a Wizard.

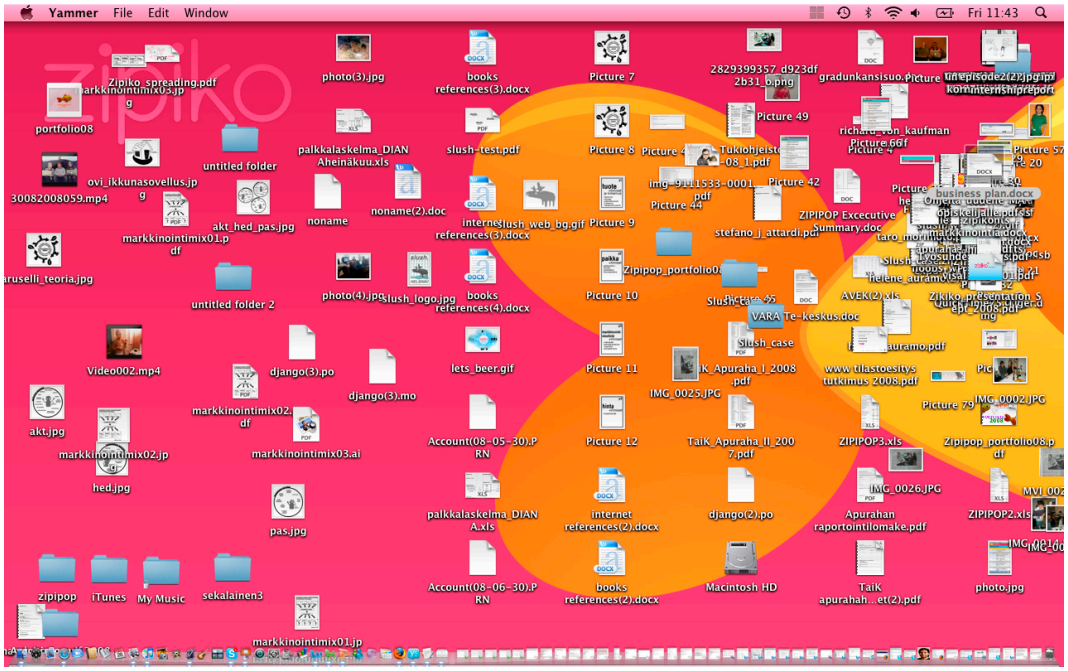


Figure 10 An example of Desktop clutter.

17 Wizards, in the context of computers, refer to semi-automated commands that require user-defined parameters in order to execute. They are often presented as a string of dialog boxes.

Not only is the desktop metaphor increasingly irrelevant, the precious space it occupies is largely wasted. The Desktop is, after all, the first screen the user sees once the operating system has launched, and thus should help the user to either continue a prior, unfinished task, or start a new one [Raskin, 2000, 31]. While it certainly is possible and convenient to place unfinished files on the Desktop and even label them accordingly, the Desktop itself does nothing to help with this tiresome process.

Lastly, the Desktop has evolved into a confusing, inconsistent mishmash of old and modern notions. From the point of view of a file system, the Desktop is nothing more than a folder that can be accessed via a file browser like any other folder. From the point of view of the user, and the user interface, the Desktop is the origo of the system — a starting point. Depending on the design, it can hold elements such as the trash can, the computer, and external media, items that are *not* to be found inside the desktop folder with a file browser.

The desktop metaphor is potent one, and its utility is useful as long as the user finds the energy to organize and maintain it. However, as it stands, the Desktop — arguably one of the most prominent elements of an OS user interface — has de-evolved into a disoriented system that, without laborious management, easily ends up as a junkyard for files, and fails to aid users in resuming or creating new tasks. While it certainly does provide a large canvas for holding a pretty picture, it does very little to promote productivity or usability.

2.2.2 Customization

Customization refers to giving users control over various preferences on how an application or user interface behaves and operates. It can also be loosely defined as a user interface design philosophy, whereby providing every imaginable option or feature is used to cover all possible use cases and satisfy all types of users [Nielsen, 1993, 15 & 121].

Customization illustrates how users' subjective preference and desire for control may contradict with their best interests. Although the possibility of customizing software is often highly regarded by users[18], in actual user tests it is shown to decrease usability and performance [Preece et al., 1994, 14–15] [Nielsen, 1993, 121–122]. To quote Raskin, "[customization] sounds nice, democratic, open-ended, and full of freedom and joy for the user, but [there appear to be no] studies that show that it increases productivity or

18 Using the broad term "users" might be somewhat misleading, as customization is generally an expert-user feature request. In fact, according to a study conducted by the Microsoft Office team, only 2% of users customized the user interface in Office 2003 — despite the fact that expert users assured the team that "everyone" customizes their Office UIs. [Harris, June 27, 2006]

improves *objective* measures of usability or learnability. Adding customization certainly makes a system more complex and more difficult to learn [Raskin, 2000, 49]."

Instead of providing extensive amounts of customization, user interfaces can lean more towards an adaptive approach. In a study conducted by Louise Barkhuus from The IT University of Copenhagen and Anind Dey from Intel Research Berkeley, users preferred context-aware features to customization, despite the lack of control they felt [Barkhuus & Dey, 2003, 8]. There is indeed a fine line between taking too much control from the user and trying to delegate as much work to the software as possible, but a subtly adaptive user interface is one way to create helpful systems that negate the need for excessive customization.

It has to be noted that adaptive interfaces pose problems of their own, such as preventing the user from utilizing his or her spatial memory to locate objects or their unexpected disappearance. An example of a failed adaptive interface concept, so-called "personalized menus", is outlined in subsection 2.4.2.

2.2.3 Feature bloat

"Feature bloat" is a term used to describe the phenomenon of features being added to software to the point where their sheer amount starts to affect usability in a negative way. Added features increase the complexity of the application: each new button, control, or instruction adds to the cognitive load of the user, and forces designers to resort to poor implementations in the user interface, such as hiding information; albeit not a problem exclusively related to GUIs, their graphical nature and heavy use of direct manipulation emphasize the negative effects of feature bloat. While feature bloat is often associated with applications, it applies to operating systems as well.

Unfortunately, feature bloat is somewhat of a necessary evil due to two reasons: user requests and marketing. As Norman points out, *users* are the ones demanding new features, not software designers [Norman, 1990, 173]. Even Microsoft Word, the well-known word processor with literarily thousands of commands, does not seem to be full-featured enough, at least according to "miles-long list of feature requests from customers" [Harris, March 31, 2006]. The other reason for promoting feature bloat is the business model for selling software; since software makers need to re-sell their software over and over again, they promote the latest version by adding (yet) more features — often based on customer feedback. Once a feature has been added, it inevitably stays in the product unless it specifically causes problems [Norman, 1990, 21].

While feature bloat is difficult to combat, there are ways to minimize it. Smart, thorough interface design coupled with strong resistance towards new features goes a long way, but modularization might ultimately be the most efficient solution. Modularization means the division of features into functional sets that can be more easily managed. [Norman, 1990, 174] Many software applications provide a limited set of often-used controls exposed at all times, and place the more obscure controls into a separate control panel that can be toggled on and off by the user. While by no means a perfect solution, it helps to keep the interface accessible to newcomers without sacrificing the total amount of features.

2.2.4 File management-related issues

Ever since the early days of computing, data has been organized into manageable chunks within a file system structure. As GUIs were introduced, the notion of files and folders emerged, partly due to their immediate understandability. Just like in real life offices, documents, spreadsheets, and the like are still stored as files, organized via folders, and placed inside a file cabinet. While sensible, these metaphors are as old as the Desktop, and equally irrelevant in the context of modern-day computing. It could be argued that users should not need to be burdened with file management and organization at all, since their focus should rather be on the content or task at hand. Nevertheless, despite efforts to eradicate[19] or even minimize it, file management has remained an unavoidable task of everyday computing.

The most glaring problem in terms of productivity (and, by extension, usability) with file management is the manual labor involved in managing files and folders. Whenever a file is created and saved by the user, it needs to be named and placed somewhere on the file system. This operation is especially irritating, since it breaks the flow of activity and forces the user to invent a unique file name [Raskin, 2000, 118]. Once saved, the file is often difficult to retrieve, especially if the name is not unique. As several usability experts have pointed out, unlike computer systems, human beings are far better at recognizing than memorizing [Nielsen, 1993, 129] [Preece et al., 1994, 118] [Raskin, 2000, 118]; current file management user interfaces, however, hardly help the user in saving and retrieving files. File retrieval has been improved by advanced, content-aware search technologies, as outlined in subsection 2.4.5, but still relies heavily on user input and memorization. Additionally, file recognition has been simplified by providing thumbnails or large previews, and providing metadata and labeling support in most modern file management user interfaces.

19 Jef Raskin's Canon Cat computer, released in 1987, completely removed files and folder structures. Instead, it used a virtually unlimited canvas for holding textual data that was navigated via issuing search strings. [Raskin, 2000, 121–122]

Despite these efforts, the fundamental problems remain largely unsolved, witnessed by the fact that file browser applications are still a crucial part of every modern GUI. Curiously, several media applications — such as iTunes or iPhoto — provide a database-like approach for file management, removing the need for using a file browser altogether[20]. This approach has also been a long-standing effort by Microsoft with their WinFS data storage and management system, based on relational databases, but has yet to come to fruition [Bangaru, 2005]. In addition to technical difficulties, it is uncertain whether humans are even willing to give all the control of their file management to software in the first place [Johnson, 1997, 172].

Another problem, albeit much more minor, is related to opening files with specific applications. Files always have one default application associated with them based on their file type — this is to ensure that files can always be opened easily. Often however, there is a need to open a file in an application that is *not* the current default, an action that currently requires the use of a contextual menu or careful drag-and-dropping. Opening the file in a "wrong" application is also a common displeasing surprise for users who are working on computers that are not their own, as defaults vary from system to system. Furthermore, files frequently require file type conversions, often for the very same reasons of easier access, another action that is not very well supported by file browsers.

Saving, retrieving, and accessing files are some of the most basic functions of a GUI system; despite their importance, current file management software provides very few truly helpful and intelligent ways to perform them — all, or at least most, of the work is left to the user. Therefore, file management quickly becomes a frustrating, daunting task that users start to avoid, complicating the situation even further. While heavy utilization of database-oriented systems might be too difficult to accomplish at the moment, there certainly are other ways to improve the current experience of file management.

2.2.5 Menus and shortcuts

A menu is another legacy user interface element that was introduced with the very first GUIs. Essentially, menus are used for grouping and hiding commands under specific categories where there is not enough space on the screen to display all the commands simultaneously.

20 It could be argued that a file browser and a visual database are very similar from the point of view of user interaction. The main difference is that visual databases can hide the underlying, often complicated file structure and present a more user-friendly method of working with files.

Over the last two decades, two potent menu standards for operating system user interfaces have been established: menu bars, and contextual pop-up menus[21]. A menu bar refers to a string of menu options visible on screen, either attached to application windows or placed in a static position, with each option expanding into a list of commands upon clicking. Menu bars can also contain small buttons instead of textual commands, often in the form of icons, with expanding pull-down menus, if necessary. Contextual menus are invoked at the position of the mouse cursor, and contain commands relevant to the item beneath the mouse cursor in a dynamic fashion — thus called "contextual". Recent innovation by the Microsoft Office team has introduced a hybrid menu bar solution called the Ribbon, utilizing a static menu bar with tabs instead of pull-down menus, coupled with contextual tabs that appear and disappear based on the currently selected element. A more detailed description of the Ribbon is presented in subsection 2.4.2.

There has been a long-standing debate between the various implementations of menus. Microsoft Windows has always attached the menu bar as a relatively small element to application windows, and encouraged the use of contextual menus, while Mac OS has retained the original, single static menu bar that is soldered onto the top of the screen instead of application windows, and advised against excessive use of contextual menus. Both approaches have their benefits and disadvantages over each other:

- The single static menu bar is consistent in its positioning, therefore easier to memorize and find. Albeit larger in size by default, it also saves screen real estate since there is only one menu bar total instead of one per application window. These benefits are especially noticeable on smaller screens.

- According to Fitt's Law, the time it takes to hit a (user interface) target is "a function of the distance to and size of the target[22] [Hale, 2007]." Because of the finite size of the computer screen, its edges and corners limit the movement of the mouse cursor: no matter how far the mouse is moved to the left, for example, the cursor stays stuck in the left edge of the screen. This in turn increases the size of any user interface targets that are positioned on the edges and corners of the screen, such as the individual pull-down menus of a static menu bar. The larger default size of the pull-down menus

21 There are certainly many other kinds of menus, such as pie menus and scroll-in menus, but these two have become most popular in terms of computer operating systems user interfaces.

22 In mathematic terms, Fitt's Law is expressed as follows: Time (to hit a target; in ms) = a + b log2 (D / S + 1), where S is the size of the (one-dimensional) target, D is the distance from the starting position to the target, and a and b are constants determined experimentally or derived from human performance parameters [Raskin, 2000, 93]. It is important to note the logarithmic part of the function, as it implies an inconstant relationship between time to hit a target and its size.

in Mac OS as opposed to Windows' menu bars also decrease the time it takes to hit them. Two properties of Fitt's Law are visualized in Figure 11.

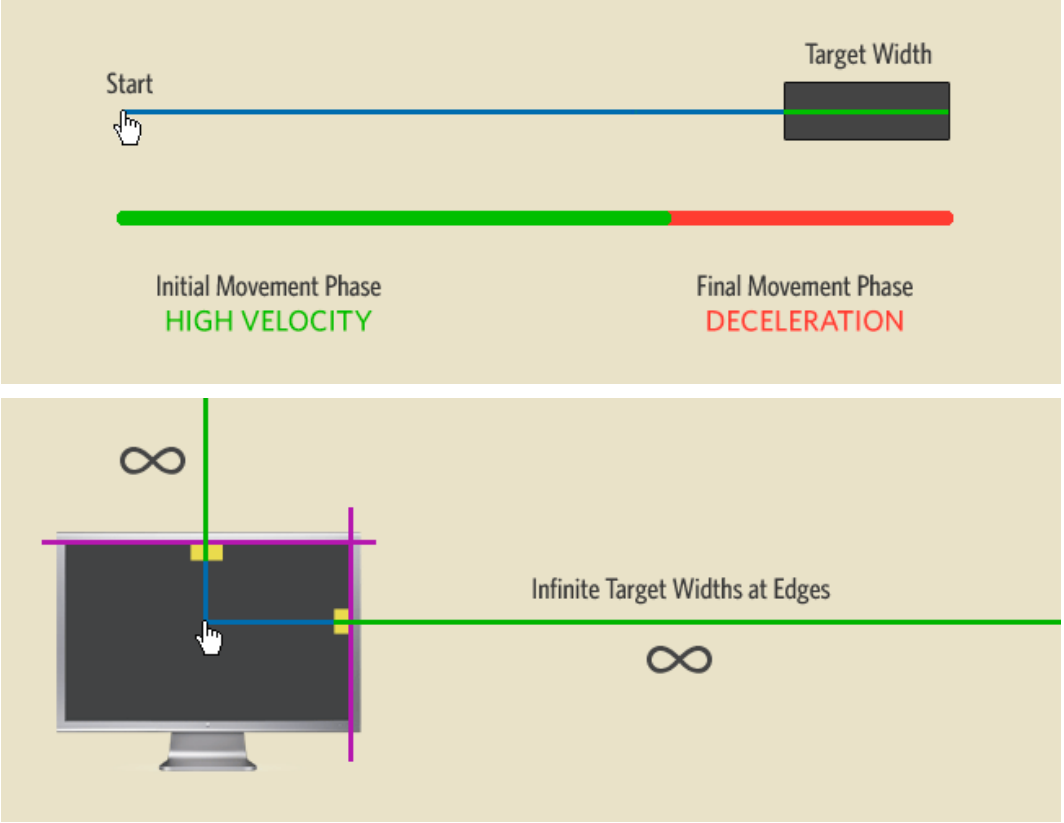


Figure 11 Two properties of Fitt's Law visualized.

- Since Fitt's law also takes into account the distance between the cursor and the target, a static menu bar soldered onto the edge of the screen decreases in performance as computers screen grow in size and resolution. Especially multi-screen setups work poorly with soldered menu bars, since the consistently positioned bar is only available on one of the screens, increasing the distance between the cursor and target immensely. Once the menu bar is visually far away from the application window, they start to feel detached from each other.

- Contextual menus, according to Fitt's law, reduce the time to issue commands considerably. They also limit the amount of commands to only the relevant ones, often a substantial reduction in complexity.

- Contextual menus are inherently invisible, and require an additional input button or key in order to be invoked. Invisibility in particular is a significant usability problem [Norman, 1990, 4] [Preece et al., 1994, 5]; the premise is quite simple, but often forgotten in user interface design: in order to convey the message across to the user

that something can be manipulated, it first has to be visible. Contextual menus have no indicators and little consistency in how they operate and where they can be used. Even if the user learns the concept of contextual menus and remembers to use them, their contents remain a mystery until invoked. They also promote the use of multi-button input devices, which in turn is problematic for interfaces that rely on finger- or pen-based input. Even in multi-button environments contextual menu-orientated user interface design can result in poor usability if certain commands are only available via a contextual menu, making them difficult to discover.

In addition to their respective disadvantages, both approaches also share common problems. The lack of visibility, again, is evident even in pull-down menus, and finding the correct command often requires tedious browsing through the various pull-down menus[23]. The situation gets worse as software grows in complexity and features are added, inevitably also increasing the amount of commands that need to reside within the menu structure. Pull-down menus are already difficult to browse and use as is, since an erroneous or inaccurate movement of the mouse cursor can switch or close the menu unexpectedly. Contextual menus, when opened by accident, cause confusion and mis-clicks since they require closing before other UI elements can be accessed again. Issuing a series of commands is slow, since executing one command closes the menu and forces the user to re-open the menu repeatedly. All these factors add up to the menu bars being error-prone, and tedious to use; contextual menus only solve part of the problem, while introducing a series of their own.

To counter the slow operation of pull-down menus, many commands have shortcuts associated with them. Shortcuts are usually evoked by holding down a combination of keys. Other shortcut methods include function keys, command name abbreviations, and the use of double-clicking [Nielsen, 1993, 41 & 139]. While shortcut key combinations are an effective way to increase the speed of issuing commands, they are hardly a perfect solution: not only are all the different key combinations difficult to memorize, they can be inconvenient to execute due to the sheer amount of modifier keys involved in the operation. There is often very little feedback on what command was executed, so an erroneous combination leading to unwanted results might go unnoticed. [Raskin, 2000, 109] A factor in making keyboard shortcuts difficult to learn is their invisibility: global, OS-level shortcuts might not be listed anywhere apart from the help files, and application-level shortcuts are hidden inside pull-down menus; contextual menus do not display keyboard shortcuts at all. In worst cases, the chosen keyboard shortcut keys have very little reference to the actual command they invoke: closing a window in Microsoft Windows is accomplished by the

23 Apple has included a menu command search field within the menu bar in their latest release of Mac OS X. It aims to help in the finding of specific menu items by name.

shortcut key combination of *ALT* and *F4*, a rather illogical choice that has no resemblance to "closing" whatsoever.

Menu and shortcut systems currently used in OS user interfaces are fundamentally excellent concepts, but simply present too much invisibility, confusion, cognitive load, and potential for errors. An improvement to issuing commands in a more effective, transparent way is sorely needed.

2.2.6 Modes

Modes in user interfaces can be defined as different states of the system in which an action issued by the user results in a dissimilar consequence. A classic example of modality is the Caps Lock key on keyboards, where the case of typed characters depends on whether or not the system is in "Caps Lock-mode" or not [Raskin, 2000, 39]; everyone who has used computers has at least once switched the Caps Lock key on by accident, only realizing it after typing a string of characters in the incorrect case. Also, the system might interpret a seemingly identical action or command by the user (such as a keyboard shortcut) differently due to being in a different mode [Nielsen, 1993, 146]. Despite the fact that modes are a significant source of user errors, they are frequently used in user interfaces [Raskin, 2000, 37] [Nielsen, 1993, 146].

Although modes are very difficult to avoid once the complexity of user interfaces grow, there are ways to minimize the risk of errors that occur because of them. One of the biggest reasons for mode errors relates to feedback, or rather the lack thereof: in the example of using the Caps Lock key, users usually do not notice the small indicator light that turns on and off when the key is pressed, mainly because it is not their focus at the time[24] [Raskin, 2000, 39–40]. Moreover, research conducted by HCI experts Abigail Sellen, Gordon Kurtenbach, and Bill Buxton has demonstrated that humans are much more likely to ignore visual feedback than other sensory feedback, such as audio [Sellen, et al., 1992].

Another way to avoid mode errors is to use quasimodes. Quasimodes are modes that must be user-maintained in order to stay active. As opposed to the modal Caps Lock key, the Shift key functions as a quasimode, since it is only active when the key is held down [Raskin, 2000, 55]. The benefit of using user-maintained modes is twofold: users make drastically

24 To counter the errors caused by accidentally pressing the Caps Lock key, many operating systems present a visual indicator — an icon or popup message — next to the input field when users are entering data in Caps Lock mode. Unfortunately, this feature often applies to password fields only. Apple implemented a threshold for pressing the Caps Lock key in their recent keyboard designs: a short, accidental press will not suffice for toggling the key.

less modal errors, and their cognitive load is reduced by not having to think about the mode of the system — increasing their performance regardless of prior expertise [Sellen, et al., 1992].

The avoidance of modality is one of the most-forgotten (or most-ignored) user interface guidelines, largely due to the perceived benefits of using modes to avoid other design problems. Unfortunately, as Murphy's Law dictates, if there is a possibility for a user error to be made because of a mode, it will inevitably occur — causing unnecessary confusion and frustration. Modal errors are also resistant to user expertise, since the natural cause of modal errors lies not in the lack of prior experience, but in the focus of attention.

2.2.7 Scrolling

Since the emergence of the Internet and browsing web pages, scrolling has become one of the most-used interactions done with a computer. Certainly, there has always been a need for scrolling, as evidenced in scroll bars being a part of the original WIMP-model, but its importance has increased considerably in the last decade. Case in point: the birth of the scroll wheel mouse and its rise in popularity.

Despite the importance of scrolling, there has been very little innovation in how scrolling is performed in software with mouse and keyboard-based systems. Scrollbars themselves are relatively small interface elements, requiring precise mouse positioning and extensive cursor movement [Raskin, 2000, 115]. The default incremental amount of scrolling performed by clicking on one of the scroll bar arrows is barely noticeable, requiring repetitive clicking or holding down the mouse button. A similar problem occurs when using a scroll wheel, where scrolling quickly becomes a tedious, repetitive finger movement; the scroll wheel is more useful for accurate, small-scale scrolling, and using it in conjunction with keyboard shortcuts for scrolling in larger steps, users can indeed quickly navigate within a window. Unfortunately, such a combined input method for scrolling is fairly demanding for most users accustomed to using the mouse alone.

Microsoft has experimented with providing a quicker way to scroll without the use of a scroll wheel in its Internet Explorer web browser. By pressing the scroll wheel (or middle mouse button if no wheel exists), users enter into a scrolling mode where scrolling is performed by moving the cursor in any desired direction. The further the cursor moves from the initial position, the faster the scrolling. While useful, the feature has its own set of problems: it is

a modal approach[25] with very little visibility attached to the most obscure "button" on the mouse — the scroll wheel.

Many trackpads on portable computers include either a dedicated part for vertical and horizontal scrolling, or enable scrolling via two-finger sliding gestures. Both work similarly to scroll bars and the scroll wheels on mice.

Another scrolling feature is the hand tool, often used in software titles that involve working on a canvas. Activating the hand tool, the cursor changes into a hand that can be used to "grab" the canvas and move it in the desired direction. Like in real life, the movement is inverted: grabbing and moving the cursor upwards will move the canvas downward, and visa versa. The same approach has been taken with touch-based devices, where scrolling is performed by a similar gesture — a slide with the finger. This inverted movement and its inconsistency with scroll bars (and scroll wheels) is most evident with touch-based devices that contain scroll bars, such as the first generations of the Nokia Internet Tablets. Web pages can be scrolled by touching the canvas and sliding with the finger, similar to a hand tool, but also by touching and sliding the scroll bar; the direction of scrolling changes, however, according to which element was touched. On Apple's iPhone, scrollbars are not manipulatable, and all scrolling is solely performed by swiping gestures.

A unified, consistent scrolling functionality for both cursor- and touch-based devices is all but called for, as are the increase of productivity and ease of use; while the finger-swiping gesture provides an intuitive way to scroll on touch-based systems, it becomes as wearisome to use as the scroll wheel when long, continuous scrolling actions are required.

2.2.8 Window management and the multitasking myth

The invention of the window metaphor is arguably the most prominent, or at least the most established, GUI innovation that came out of the research done at Xerox PARC. Windows are essentially visual containers, or frames, that hold within them any kind of data — much like real life frames can encompass photos, paintings, prints, et cetera.

25 The feature can also be used as a quasimode, by holding down the scroll wheel as long as the scroll action is needed. Unfortunately, a scroll wheel is not the most ergonomic design for pressing and holding, rendering the quasimode approach too uncomfortable to use on a regular basis. Also, since the option of using both a mode and a quasimode is provided, users who discover the feature by accidentally pressing the scroll wheel might never think of trying out a press-*and*-hold action.

In modern operating systems, windows can generally be manipulated in several ways with the help of user interface buttons, handles, and scroll bars. Typical operations include moving and resizing the window, or scrolling its content. When operating systems started supporting multiple windows and multitasking, and users began to take advantage of those features, windows started to require separate management. Operations such as maximizing the window to the size of the entire screen, or minimizing the window in order to "put it away" have been added to help users focus on specific windows. More advanced features such as Apple's Exposé and the many incarnations of Virtual Desktops (presented in subsections 2.4.3 and 2.4.8, respectively) have provided even more sophisticated and complex tools for organizing and accessing windows. The addition of these more advanced features proves that many users feel the need for arranging their windows — and for good reason; with operating systems now easily handling dozens of windows without any noticeable slowdown, it is hardly a wonder that workspaces are filled with ever-exceeding amount of application windows. A popular action to sidestep this problem is to simply maximize a window to fit the whole screen, effectively hiding the compiled disarray of other windows underneath it.

There are other issues, as well. Occluding windows tend to hide parts or even complete windows behind them, making it difficult to see and switch between windows directly; comparing two documents is a daunting task, often requiring precise resizing and repositioning of both windows[26]. Even recognizing the active window can be challenging, especially on Mac OS X where an *application* can be active even when it does not contain any windows per se. To make matters even more confusing, applications can also contain panels, visually window-like elements that are not treated as such technically. Dialogs, preference panes, and status messages are special windows that can cause perplexity because of their window-like nature: just like any window, they can be moved, occluded, and hidden behind other windows and thus become disassociated with their parent applications or windows[27]. Another problem is the seemingly random position windows are placed upon opening or creating them [Preece, et al., 1994, 303]; a common practice is to place new windows in the top-left corner of the screen, but this can introduce extra cursor movement for users with larger screens — or even worse, multi-display setups.

---

26 A little-known feature in Windows is the automatic tiling of all visible windows, available only via a contextual menu from the taskbar. It is excellent for resizing and setting up windows next to each other without any occlusion, but the implementation is not good enough for frequent use; resizing is applied to all visible windows, and their prior positions and sizes can not be reapplied automatically.

27 A wonderful user interface element, the Sheet, has replaced some of these windows on Mac OS X. Sheets are attached to their parent windows, and can therefore never be disconnected with them: if the parent window is moved, resized, or hidden, so are the sheets.

A typical user error related to window management is the unexpected and unintended change of the active window, or focus. Everyone must have experienced the frustration of typing in a word processor, for example, when the window focus suddenly switches to a pop-up dialog window issued by another application or the operating system, only to be accidentally dismissed, accepted, or otherwise manipulated by the string of characters originally meant to be typed into the word processor instead. While most user interface implementations require clicking on a window in order to make it active, some provide the possibility to change window focus by simply moving the mouse cursor on top of windows [Preece, et al., 1994, 300]. There is also the concept of click-through — being able to manipulate inactive window interface elements without first making the window active. Needless to say, inconsistent use of click-through will ultimately lead to user errors.

A fundamental issue related to window management revolves around multitasking. Multitasking, carrying out several tasks at the same time, is common human behavior — in fact humans are multitasking more often than performing tasks in a serial manner [Preece, et al., 1994, 105]. The need for multitasking is largely satisfied with the possibility of opening several windows each representing a separate task. However, discounting automatic cognitive processes such as walking or breathing, humans are not quite as good at multitasking as we like to believe. Not only are humans prone to distractions, but can also execute only on *one* focus and thought-demanding task at a given time [Raskin, 2006]. Certainly, it might not be challenging to switch from one task to another and back, but zeroing in on both at the same time is impossible. This has much to do with the way human eyes and brain co-operate: while it is possible to redirect attention to objects within a single fixation, tasks such as reading are only possible at the very point of focal fixation [Ware, 2004, 146]; while humans have a pair of eyes, they still focus on one point, not two. With such a limited field of focus and attention, humans do not benefit from user interfaces that promote multitasking as opposed to completing tasks in a sequential order. Multi-window user interfaces as they are presented in modern operating systems increase visual clutter and cognitive load; it is no accident that many users prefer to maximize their windows to fill up the entire screen, since it is one way to help focus on the given task.

The flexible and venerable window metaphor has been a cornerstone of the WIMP model since its introduction. It does not come without its problems, however, as the ever-growing amount of windows, panels, and dialogs lead to confusion and unintentional errors. The very notion of overlapping windows introduces constant window management and can even cause windows to disappear unexpectedly. While multitasking is promoted, the model of simply layering windows on top of each other surely cannot be the most helpful solution, even when enhanced with more advanced management features such as Exposé or Virtual Desktops.

2.3 Alternative OS user interfaces

To counter some of the problems posed by traditional, WIMP-based OS user interface designs, several alternative approaches and experimental projects have emerged. Many of them are still under development.

It is of interest to note that none of these alternative OS user interfaces have a large presence on the market, either because they are run by small development teams — sometimes even by individuals — and lacking the big marketing and R&D budgets of major corporations, or because they are not finished, working products. Some have simply failed to gain enough popularity due to their shortcomings.

Command line interfaces are certainly an alternative to GUIs, often solving or circumventing many of their problems. Since CLIs can be found integrated into modern operating systems, they are not listed here as a separate alternative. Instead, their benefits are mentioned where relevant.

This list is by no means a complete summary of all the available alternative OS user interface designs; it merely serves to illustrate some of the different directions being explored and provides a broad overview of the current range of designs. There are valuable lessons to be learned from the advantages and disadvantages of each design approach. The list is presented in alphabetical order.

2.3.1 Archy

Archy, as illustrated in Figure 12, is Jef Raskin's "prototype computing environment based on years of research on this" [Raskin Center for Humane Interfaces, 2007], maintained by the Raskin Center for Humane Interfaces. Archy trades the traditional window and the desktop metaphors with a virtually unlimited, text-based canvas that contains all the users data, and is controlled by textual commands. There are no applications or files — users simply "leap" from content to content via search methods. Future revisions of Archy plan to provide a zooming metaphor for inspecting data and following hyperlinks.

Archy is a radical departure from the WIMP model. While it achieves, among other things, unlimited undo steps, eradication of modes, consistent data manipulation, removal of the file/folder metaphor, and prevention of data loss, it lacks all of the visual attractiveness of current WIMP-based systems; Archy is, atypically, an example of function over form. It is also difficult to imagine that a system without applications would be favorably welcomed

by 3rd-party software developers[28]. Nevertheless, in addition to its usability feats, Archy presents interesting concepts such as an enhanced command line interface, and the notion of zooming in and out of objects.

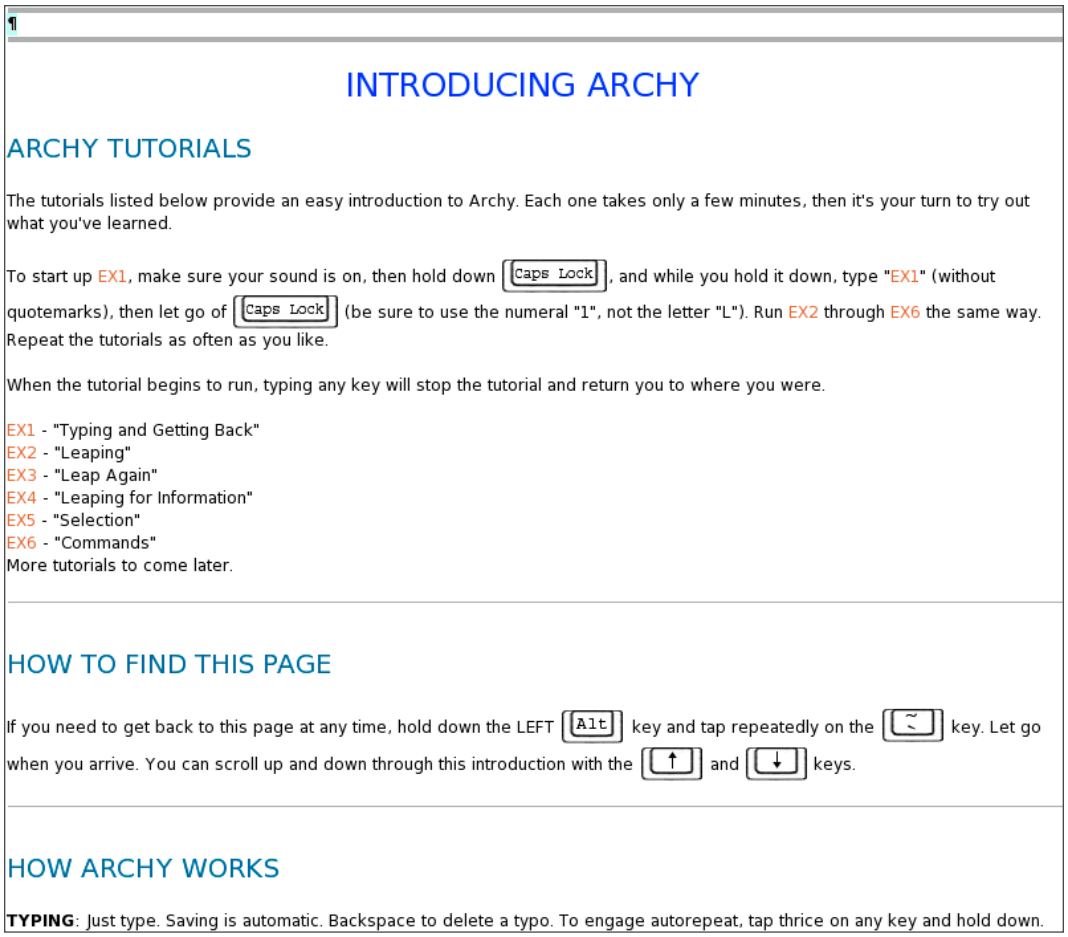


Figure 12 Jef Raskin's Archy operating system.

2.3.2 BumpTop

BumpTop, as presented in Figure 13, is not a complete OS user interface redesign, but rather a novel approach to expand on the desktop metaphor. Co-created by Anand Agarawala and Ravin Balakrishnan, BumpTop presents a 3-dimensional Desktop, or room, where files are represented as small square-shaped chips. There are numerous gesture-based actions the users can use to organize and manipulate the chips. Included is also a physics engine that provides an "engaging, vivid user experience [Agarawala & Balakrishnan, 2007]."

28 Raskin proposes that 3rd-party software developers would sell sets of commands (or functions) instead of applications that would directly integrate into the operating system architecture. [Raskin, 2000, 144]

While BumpTop does not solve any of the problems OS user interfaces face, expect perhaps the lack of organization tools for the Desktop, the immense media and public interest in it proves that attractive and engaging user interfaces indeed get noticed and are able to generate excitement. In addition, the design utilizes such contemporary concepts as pie menus and gesture-based interaction.

Figure 13 The BumpTop Desktop in action.

It is recommended to see a video demonstration[29] of BumpTop, since most of its attractiveness stems from the real-time animation and physics engine.

2.3.3 Microsoft Bob

In 1995, Microsoft Corporation released a software package entitled Microsoft Bob that was designed to provide a more approachable user interface for non-technical users. Microsoft Bob, as illustrated in Figure 14, featured a friendly, cartoon-like environment that resembled a house with several rooms the users could explore and use for launching applications, as well as a selection of animated assistants similar to the Office Assistants

29 A video demonstration of BumpTop can be found on the video-sharing web service YouTube at http://www.youtube.com/watch?v=MoODskdEPnQ.

later introduced in Microsoft Word and Microsoft Windows XP[30]. Technically, Microsoft Bob was an application that ran on top of Windows, not replacing the underlying operating system per se.

Microsoft Bob was quickly deemed as a failure by the market, and was never developed beyond the initial release. While the product certainly was approachable, the benefits were largely undermined by the system's inability to adapt to the users' increasing expertise. Microsoft Bob illustrates the problems faced in mimicking real-life environments and agent behavior in user interfaces, and as such provides insight for designing adaptive and helpful computer systems.

Other room-based operating system GUIs include the Packard Bell Navigator, developed between 1993 and 1996, PDA operating system Magic Cap by General Magic, and the more recent Microsoft Research "test bed" [Microsoft Research, 2005] — Task Gallery.

Figure 14 Microsoft's Bob user interface.

30 The animated dog that helps users find files on Windows XP first appeared in Bob as an assistant named Rover.

2.3.4 Project Elevate

Project Elevate, as presented in Figure 15, is an alternative OS user interface for Linux designed and developed solely by Kostis Kapelonis. He argues for simpler, adaptive, and discoverable interfaces that let the users focus on tasks instead of applications. [Kapelonis, 2007]

Project Elevate consists of four distinct applications, a central hub (or launcher), an index-based file vault, a vaguely described window management interface, and an interface for external media. While Kapelonis does not provide much detailed information on exactly how Project Elevate intends to remedy problems of current WIMP-based systems, his focus on usability and goal of rethinking the OS user interface model are noteworthy.

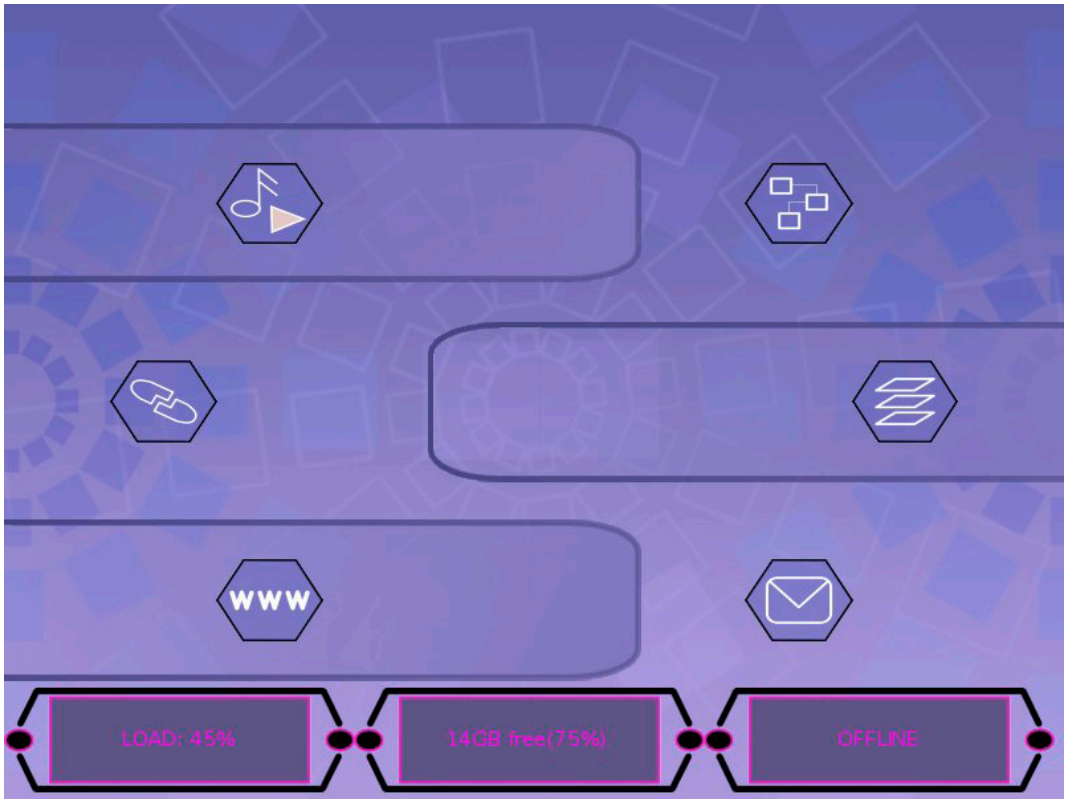


Figure 15 Project Elevate includes a central hub for launching applications.

2.3.5 Project Looking Glass

Project Looking Glass, as shown in Figure 16, is an "open platform to explore innovative 3D user interface [sic]" [Sun Microsystems, 2007] by Sun Microsystems. The project does not aim to replace the existing WIMP model; instead, it encloses the model in a more

"accurate"[31] 3-dimensional environment. Windows can be rotated freely around their axes, turned around, and scaled.

The benefits of using a 3D environment to manipulate data that is essentially 2-dimensional are questionable; Project Looking Glass has not yet demonstrated any usability improvements over 2D OS interfaces. The most interesting aspect is the open-source development approach, albeit it has yet to produce any meaningful results.

Figure 16 Sun Microsystem's Project Looking Glass GUI.

2.3.6 SymphonyOS / Mezzo Desktop Environment

SymphonyOS, as presented in Figure 17, is a Linux-based operating system managed by Ryan Quinn with an untraditional user interface, the Mezzo Desktop Environment [Quinn, 2007]. Mezzo is the creation of Jason Spisak, who has also published his guidelines for the user interface. According to Spisak, Mezzo focuses on usability and making computers

31 Many windowing systems are in fact 3D environments where windows have depth values, even in technical terms. After all, overlapping, occluding windows would not be possible in a strictly 2D environment. Project Looking Glass, however, uses the third dimension more elaborately than traditional WIMP interfaces, thus can be considered as a more "accurate" 3D interface.

easier to use. As he explains in his guidelines: “I’ve never seen a more complicated tool [...] than today’s PCs. To this end, they need to be simplified. They need less clicks, less hidden functions, less menus, and less visual trickery [Spisak, 2005, 23].”

Mezzo places common items of interest, physical devices, programs, documents, and the trash can in the four corners of the screen where they are most easily accessed according to Fitt’s law. These target items, when clicked, open up lists of related tasks, applications, documents, et cetera, often based on their popularity. The desktop metaphor is omitted; instead, the screen real estate is used for window thumbnails for application switching.

While there are many novel and thoughtful user interface improvements in Mezzo — such as the replacement of the Desktop with a more organized, adaptive, and simpler concept — the design falls short once application windows are introduced. While Spisak argues against nested menus and scrolling [Spisak, 2005, 6-7], no solutions are provided for replacing the menu bar or reducing the inconveniences of scrolling. There are also no tools to help with window management.

Nevertheless, Mezzo is one of the most inspired OS user interface redesigns on the market. The design is clearly focused on usability while still managing to retain much of the familiarity of the traditional WIMP-based systems.

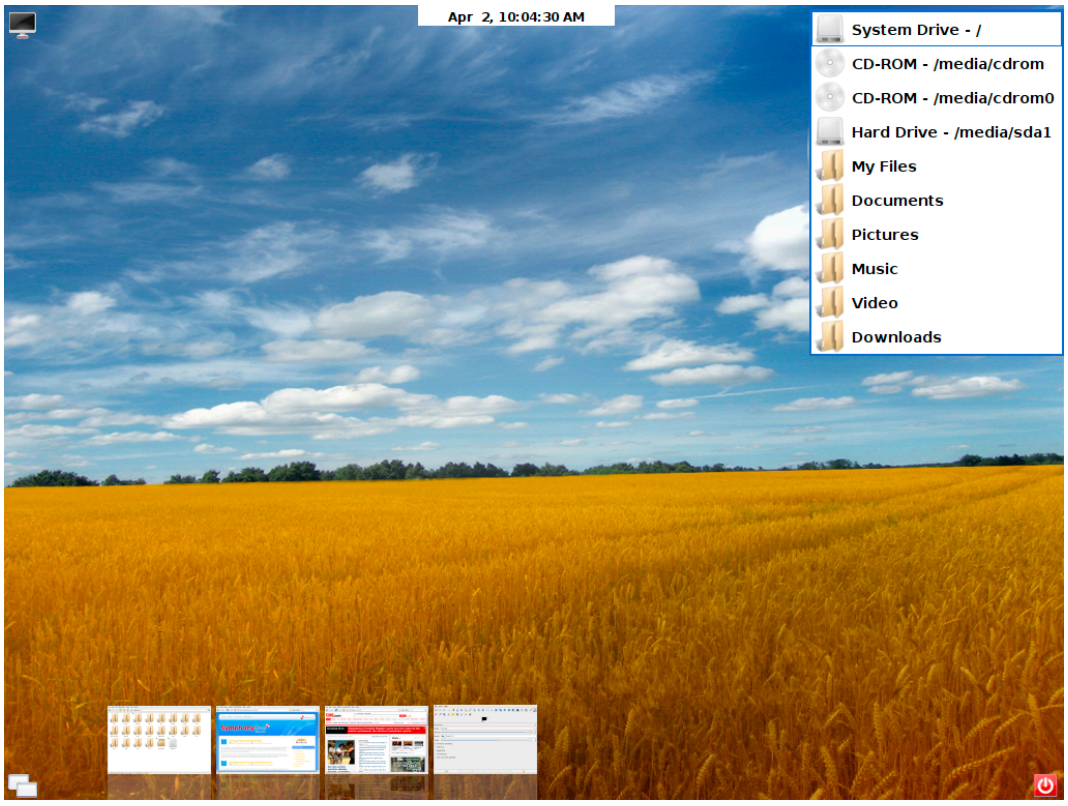


Figure 17 The Mezzo Desktop Environment in SymphonyOS.

2.4 Innovative additions to WIMP-based OS user interfaces

In addition to the more experimental, norm-breaking alternative OS user interfaces, there have been numerous smaller innovative additions to the traditional WIMP model that have tried to resolve some of the problems of its implementations. Albeit not necessarily radical, many of these innovations have changed the overall computing experience enough to warrant a mention.

The compiled list is not a complete summary of all the innovative OS user interface additions, rather merely an overview of the most relevant and interesting concepts. It is also worth noting that none of the examples are purely technical innovations, although several certainly would not have been possible to create without the advancements of software and hardware engineering. The list is presented in alphabetical order.

2.4.1 Dock

The Dock first appeared in the Arthur operating system released as early as 1987, and was later popularized by NeXTSTEP and Mac OS X in the following decades [Holwerda, 2007]. The Dock in Mac OS X could be characterized as a hybrid between an application launcher and switcher; the user can freely customize the dock by adding application and file icons to the dock for global, single-click launching. In addition, the dock contains all running applications, allowing for easy application switching. For decreased Desktop clutter, shortcuts and files can instead be placed in the Dock. While other operating systems that contain a Dock concept, such as RISC OS, Ubuntu[32], CDE[33], the XFCE Desktop Environment for *NIX systems[34], Project Looking Glass, and others all have their own unique version of the Dock with a varying feature set, the basic concept remains the same across all implementations. Microsoft Windows has long had a similar concept, the Quick Launch bar located inside the Taskbar, but this feature is optional and only one of many ways to launch processes in Windows.

The implementation of the Dock in Mac OS X has received much criticism from many usability experts, including Bruce Tognazzini, an ex-employee of Apple Computer, Inc.

32 Ubuntu is a distribution of the Linux operating system, based on Debian.

33 CDE, or Common Desktop Environment, is a proprietary desktop environment for Unix, used primarily by Hewlett-Packard and Sun Microsystems.

34 Xfce is a free desktop environment for Unix (and Unix-like platforms), originally similar to CDE.

who founded the Apple Human Interface Group, and is currently working for the Nielsen Norman Group. In his article *Top ~~Ten~~ Nine Reasons the Apple Dock Still Sucks,* Tognazzini outlines the major problems related to the Dock. Although his article is specifically targeted at the Dock in OS X, many of the criticisms apply to the general concept as well: the large size of the Dock, the lack of labeling, and the identical file and folder icons. [Tognazzini, 2004] In addition, the Dock concept has one major drawback: it requires management. Since the dock can never hold all the applications of a system, users are required to add and replace favorite applications from the Dock on a regular basis. Over time, this process becomes so tedious that users either place as many icons in the Dock as possible in order to avoid management, or neglect managing the Dock altogether. Since the Dock is most useful when presenting only a handful of favorite or current applications, both results are impractical and counterproductive.

Figure 18 presents various incarnations of the Dock.

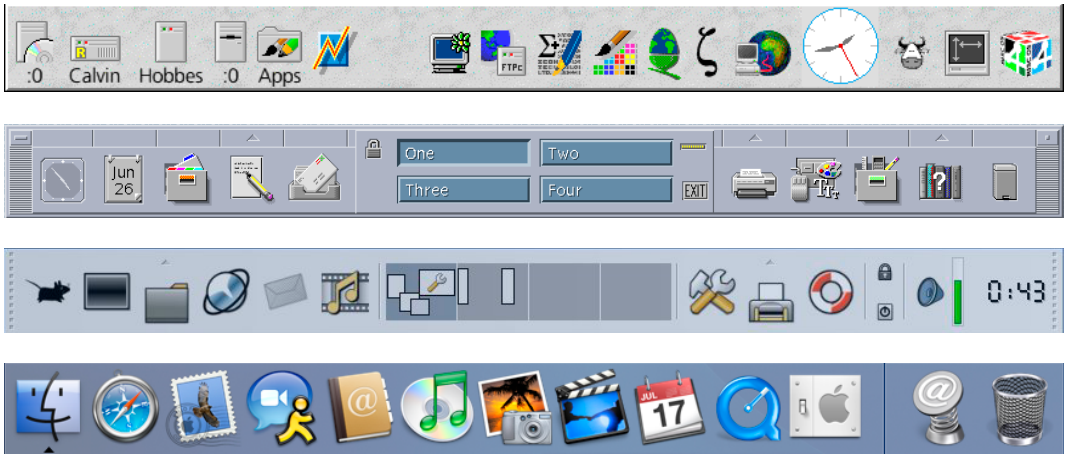


Figure 18 From top to bottom: RISC OS, CDE, XFCE, and Mac OS X Dock designs.

2.4.2 Dynamic menus

Dynamic menu is a general, wide-ranging term for menus and menu-like interface elements that have their contents changed either according to the popularity of menu items, or based on the current selection. Essentially, dynamic menus aim to increase productivity by counteracting the growing size of pull-down menus, or reusing the space needed by user interface panels such as toolbars.

One of the most memorable and radical uses of dynamic menus was conducted by Microsoft in the Start-menu of its Windows operating system as well as in the application menus of its Office productivity suite. The feature, as illustrated in Figure 19, was called “Personalized Menus” or “IntelliMenus”, and was an attempt to reduce the size of menu lists by showing

the most popular commands only [Harris, 2006]. As the user favored certain commands over others, the contents of the menus changed accordingly. The complete list of commands was available via a button at the bottom of the menu.

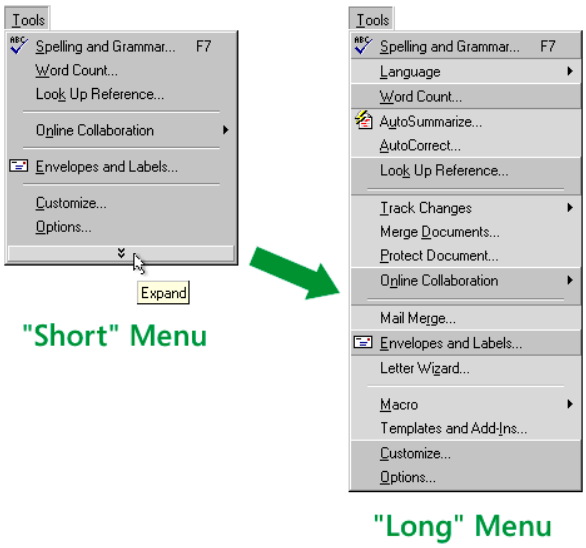


Figure 19 A Microsoft Word adaptive menu in both its "short" and "long" forms.

According to Jensen Harris, the Group Program Manager of the Microsoft Office User Experience Team, Personalized Menus were not successful: the default-compiled list of commands was never personal since the user had not started using the application yet, scanning of menu items always took two steps instead of one since the menus had to be expanded first, and the auto-personalization algorithm introduced unpredictability to the menus. Similar results were obtained from another, similar feature that dealt with adaptive toolbars and their overflowing icons. [Harris, March 31, 2006]

In Office version 12 (released in 2007), the Microsoft Office User Experience Team opted for another solution, the so-called "Ribbon" interface. The Ribbon is essentially a large toolbar that consists of several tabs — each containing commands related to the specific tab. Selecting objects, such as pictures or tables, adds new tabs to the Ribbon that contain commands related to that object type. When the object is deselected, the additional tabs disappear since they would have been disabled regardless. [Harris, 2005] [Harris, March 7, 2006]

Prior to the user interface changes in Office 12, Microsoft experimented with another dynamic menu style, the "split menu". The split menu retains the alphabetical list style, but duplicates recently used items at the top of the list for faster access. According to some research, split menus improve productivity significantly [Selvidge, 2002].

Yet another approach has been taken with the Windows Start-menu in Windows XP and Windows Vista, where recent and popular applications are duplicated in a separate column. Similar dynamic lists based on recently accessed items can be found in many operating systems and application menus — the "Open recent"-file menu option being one of the most prominent examples.

Adaptive and dynamic menus present a typical user interface problem. While there are clear theoretical — and in some cases practical — benefits in using them, users tend to prefer static menus regardless [Selvidge, 2002]. Microsoft's experiments with more radical dynamic menus (and user agents[35]) has exposed the dangerous nature of adaptive user interfaces, thus they must be used with utmost caution.

2.4.3 Exposé

Exposé, as shown in Figure 20, is a window-management feature first introduced in the Apple Mac OS X 10.3 operating system. Exposé provides three distinct ways to temporarily organize windows, either scaling all visible windows down to fit on the screen without overlapping each other, scaling only the windows of the currently active application, or wiping all the visible windows off-screen for clear access to the Desktop [Apple Support, 2007]. The feature has later been added to other window managers and compositors, such as Compiz (running on top of the X Window System).

Figure 20 Exposé in action.

As a window switcher, Exposé is remarkably intuitive; it provides relatively large, "live" previews of all the visible windows coupled with smooth animation to help the user understand how the various windows relate to each other spatially. Exposé also retains the relative sizes of windows, and as such comes very close to being a zooming interface.

While Exposé is arguably the most easy-to-understand window switcher, it is not without its problems. Scaled windows are not labeled unless hovered-over with the cursor, making distinction between similar windows difficult. Despite their resemblance to full-size windows, "exposéd" windows cannot be manipulated in a similar manner — they cannot be moved, resized, or minimized. Since Exposé only works for visible windows, any windows that have been minimized or hidden, as well as any inactive tabs will stay invisible, contra-

35 The infamous "Clippy" user agent helper, also known as "Office Assistant", that appeared in Microsoft Office versions 97-2007 is a wonderful example of the dangers of adaptive interface elements.

dicting the notion that Exposé can be used as a dependable window switcher. Furthermore, the distribution of scaled-down windows seems arbitrary: not only does Exposé omit the stacking order of application windows, it also places windows in seemingly random positions when activated, removing any help from the users' spatial memory.

Exposé is also a prime example of the burden of legacy. While it is a genuinely useful feature, it has been added *on top* of the existing window management of Mac OS X instead of being engineered as a central part of the computing experience[36]. Thus, it remains a novel, yet ultimately not a pivotal feature.

2.4.4 Gestures

Gestures refer to specific combinations of input device movement and clicking that trigger an action in the software. Using a mouse, a gesture might be activated via holding down a mouse button, moving the cursor to a given direction, and finally releasing the button — a combination that further initiates a command; one of the very early implementations of gestures was the drag-and-drop metaphor on GUI operating systems, utilizing such a combination of mouse actions. Recently, gestures have become an increasingly interesting topic with the introduction of touch-based user interfaces found on mobile devices such as PDAs, Internet tablets, and mobile phones that due to their small screen sizes benefit from the removal of traditional user interface elements. One of the best examples of the use of gestures in traditional, cursor-based software is the Opera web browser, which lets the user execute "frequently performed browse operations with small, quick mouse movements

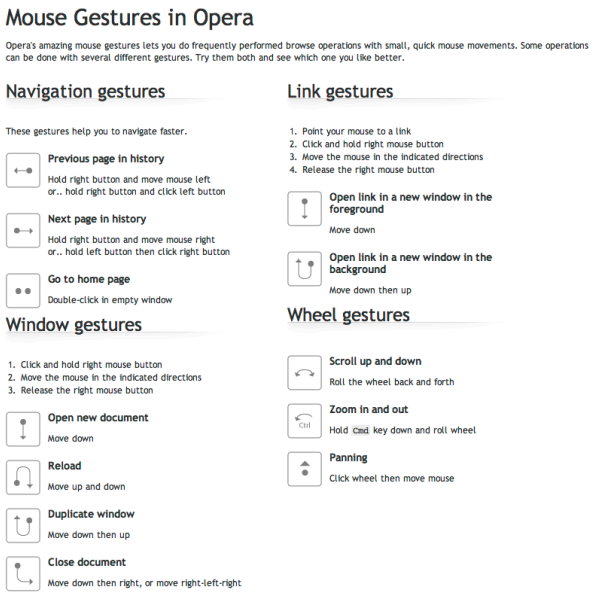


Figure 21 Mouse gestures supported in the Opera web browser.

36 In recent years, Apple has made Exposé more accessible by enabling its activation via extra mouse buttons, screen corners, a Dock icon, and even by redesigning their keyboards for easier hardware-key shortcuts. However, despite adding accessibility, Exposé still remains a non-fundamental feature.

[Opera Software].” Figure 21 presents an overview of the supported mouse gestures in Opera.

The main benefit of gestures is the possibility to execute them anywhere, such as scrolling a list without moving the mouse cursor onto a scroll bar, and not using keyboard shortcuts. While gestures can improve the speed of executing commands, there are some serious drawbacks, as well. Gestures are invisible by nature, making them difficult for users to discover without the use of additional reminders or help. In addition, gestures have to be learned and memorized, introducing similar cognitive load as keyboard shortcuts. There is also a high risk of user error, since gestures might either not be recognized correctly by the system, or require near-perfect precision in order to be recognized at all. The lack of feedback is another, related problem.

2.4.5 Search and metadata

With the rise of the Internet, and due to the ever-increasing amount of files and folders on personal computers, searching has become a central part of conventional computer usage. While not exactly an innovation in *user interfaces* per se, the paradigm shift from browsing to searching (when more useful) has led to notable changes in user interface design. Beside the typical file name search, several recent operating systems such as Mac OS X, Windows Vista, and SUSE Linux 10[37] have introduced searching as a supplementary option for launching applications. This has been made possible by advancements in search technologies, since it is now possible to find a specific file in a fraction of a second even amongst hundreds of thousands of files.

The utility of searching has also been improved with the increasing use of metadata. Metadata is information about information, or in practical terms, additional information about files that may or may not be obvious or useful to the end-user. Examples of metadata associated with files include their author, creation date, keywords, rating, et cetera. Metadata enables more sophisticated search queries that are not based solely on file names. It is also a cornerstone of databases, which sometimes are solely populated by metadata.

Search has also been enhanced with “typed commands” — text-based actions similar to menu-items that enable the user to issue specific commands without a mouse (or similar input device), much like command line interfaces. Typed commands are described in more detail in subsection 2.4.7.

37 The SUSE Linux 10 desktop operating system is a product owned and distributed by Novell. SUSE is being offered both as an enterprise and open version.

2.4.6 Tabs

Tabs could be defined as windows inside windows; whereas applications can contain multiple windows inside their workspace, tabs enable a compilation of individual windows inside a window framework — a rather confusing concept despite its acceptance. First introduced by IBM in the late 1980s, and later popularized by Internet browsers, tabs provide a more organized approach to manage multiple windows, and enable fast switching between them [Holwerda, 2008].

As a user interface concept, tabs effectively reduce the amount of windows presented on screen, since they combine several application windows inside one window. While a wonderful feature, tabs integrate very poorly with windows because they are not treated as such by the user interface itself. While switching between tabs within one application might be convenient, switching between tabs of various applications is much more complicated. Individual tabs cannot be resized or moved, since they are attached to the parent window. Furthermore, tabs are generally identified by their names, not their content, requiring the memorization of page titles instead of providing the user the possibility to recognize pages based on their appearance[38]. Most importantly, the various window management tools provided for windows do not support tabs well enough in order to make them feel and behave like an integrated part of the system.

Despite the theoretical disadvantages of tabs, it could be argued that they have been a well-received addition to modern GUIs. For example Thorn Holwerda, an editor for OSNews.com, claims that "out in the real world, tabs have been a smashing success. Whether you are dealing with an advanced user, or the hypothetical grandmother, few will have problems understanding the concept of tabs, and how to properly use them." [Holwerda, 2008]

Figure 22 presents an example of tab usage.

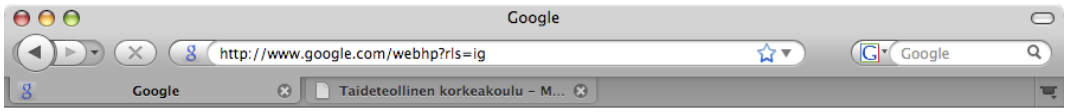


Figure 22 Tabs implemented in the Firefox web browser.

2.4.7 Typed commands

In the early days of personal computing, before graphical user interfaces appeared on the market, operating systems were controlled solely by typed commands executed with a

38 Omniweb, a Mac OS X web browser, introduced the concept of presenting tabs as thumbnails of the actual web pages. This concept was later used in Internet Explorer 7.

command line interface. Although GUIs have since provided a more visual, mouse-based method of interacting with the computer, most operating systems still contain some form of a CLI terminal, either as an integrated part of the system or as a separate application. It could be argued, in fact, that features such as pull-down menus in GUIs are essentially just a compilation of available typed commands [Johnson, Interface Culture, 151].

Since the notion of typed commands has long prevailed in the history of computing, and their benefits over a completely graphical user interface environment have long been known, there have been few attempts to seamlessly integrate the concept of typed commands into GUIs or make them more accessible to users who have previously only used point-and-clicking for inputting commands. Enso, created by Humanized — a small software company dedicated to building more "humane software" [DiCarlo, et al., 2007] — is one example of such an attempt; it adds an integrated, plug-in based environment to recent Microsoft Windows operating systems for inserting typed commands. Users can invoke Enso at any time and then type commands such as "open Notepad", "define dexterity", or "calculate 165+58" in order to execute those commands.

In recent years, typed commands have been integrated to many search engines. Google and Apple have provided similar functionality integrated in their respective search engine technologies. Apple is also working on bringing many of the benefits of CLIs, such as batch processing, to their GUI with the help of dedicated scripting applications [Apple Support, 2007].

2.4.8 Virtual Desktops

Virtual Desktops is a software feature for multiplying Desktop workspace by providing more than one screen for holding application windows. The name is actually misleading, since in current implementations of Virtual Desktops the Desktop itself — and all icons it contains — remain fixed when switching between the various "Desktops"; the goal of Virtual Desktops is not to provide more space for icons, but rather to present more application window management functionality and, ultimately, to reduce window clutter.

The concept of Virtual Desktops emerged already in the mid-1980s. In 1986, Xerox PARC researchers published an article titled *Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface*, which described the functionality of dividing the user's workspace into a suite of virtual spaces. [Henderson, Jr. & Card, 1986] Since then, there have been several implementations of Virtual Desktops, and nearly all modern operating systems either have one built-in or can be supplemented by third party Virtual Desktop applications; especially various Linux window managers as well as CDE-powered UNIX systems that run on the X Window System have had a

long tradition of Virtual Desktops. Recent advancements in compositing managers[39] have induced a host of innovative implementations for Virtual Desktops, such as grids, stripes, carousels, and even 3-dimensional cubes. Figure 23 shows various implementations of Virtual Desktops.

Although window clutter is a common problem of WIMP-based user interfaces, Virtual Desktops have remained a feature only so-called expert computer users tend to utilize. Window management is tiresome as is, and manually moving windows from one Virtual Desktop to another quickly becomes a tedious process. There is also a very fundamental problem of logic: even though it sounds as if the feature provides multiple Desktops, instead the Desktop remains completely static and only the application windows change. Apple tried to solve this problem in their implementation of Virtual Desktops by calling them “spaces”, but the underlying illogical rationale remains nevertheless.

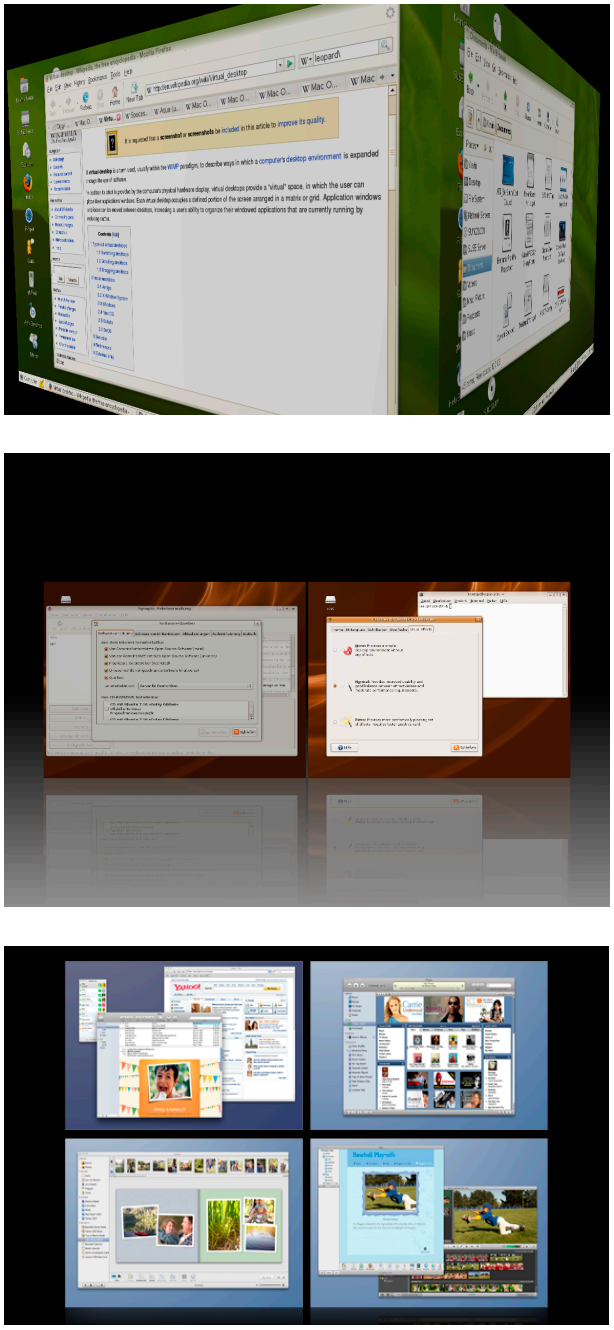


Figure 23 **From top to bottom: implementations of Virtual Desktops in openSUSE, Ubuntu, and Mac OS X.**

39 A compositing manager is responsible for drawing and presenting elements in a GUI [Microsoft Developer Network, 2007]. Modern compositing managers such as Microsoft WDM, Apple Quartz, or open-source Compiz enable advanced graphical effects such as transparencies, reflections, and blur to be applied to application windows and other elements in a GUI.

2.4.9 Widgets

Confusingly named[40], Widgets (or Gadgets, as Microsoft calls them) are simple mini-applications that often use Internet technologies for their backend and/or representation. Their main purpose is to present web-based information quickly and accessibly in a small form factor; common Widgets include clocks, newsreaders, status indicators, weather reports, et cetera. Figure 24 presents various Widget implementations.

The notion of Widgets originated from the lack of true multitasking support in early graphical operating systems. The first version of the Macintosh OS included what was called "Desk Accessories", small sidekick applications with specific, practical functionality that complimented larger applications. Similar solutions were also provided for DOS and GEM. [Holwerda, 2007] With the advent of the Internet, Widgets were reborn as web-powered applets and integrated into operating systems and web browsers. Due to their simplistic nature and low-cost development, the amount of available Widgets has exploded into the thousands across the various implementations.

Figure 24 **From top to bottom: implementations of Widgets in Yahoo! Widgets (under Windows XP), Mac OS X, and Windows Vista.**

40 The term "widget" has traditionally been used to represent a user-controllable UI element such as buttons, pull-down menus, scrollbars, et cetera.

While not a necessity, Widgets provide quick access to commonly used functionality and information; as such, their usefulness is dependent on how frequently their functionality is required. In terms of UI design, Widget platforms (the foundation on which the various Widgets run) are as varied as the Widgets themselves: some mix Widgets along with other applications, others present them in a separate layer of their own, and the rest provide a mix between the two. Widget user interfaces present perhaps the most chaotic and inconsistent mesh of designs within the field, as virtually every single Widget is unique in look-and-feel.

3 Proposed interface solutions and initial mockups

This chapter introduces the proposed interface solutions in regard to the context and problems discussed in the prior chapter. The most important design decisions are outlined according to the scope of this paper.

In addition, the chapter is illustrated with initial mockups that aid in visualizing and understanding the various design solutions. Because of the equal focus on aesthetic and functional design of the proposed interface solution, the look-and-feel of the user interface is of high importance. Although the initial mockups presented in this paper have been created with great care, they do not necessarily represent the optimal, final look-and-feel for the user interface.

Most mockups are presented in a screen resolution of 1024x640 pixels, a reasonably small widescreen resolution that is close to the minimum specified resolution of 800x480 pixels.

3.1 General design concepts

There are several overreaching design concepts within the proposed interface solution not associated with any one particular issue or resolution. These general design concepts are outlined in this section.

As the project is not being developed as a commercial product, the importance of branding is moderate at best. However, naming the proposed interface solution is in order for the sake of readability: for the rest of the document, it will be entitled *Stripes*. The name seems friendly, and correlates well with its stripe-oriented visual design.

The standing design of *Stripes* was a result of several months of brainstorming, and many of the current design concepts have been through several iterations. Appendix A presents some of the early paper mockups that illustrate the creative process leading to the particular version presented in this paper.

3.1.1 Visual look-and-feel: luminance and the use of color

Although the visual look-and-feel of *Stripes* is not based on any existing guideline document, and the visual style presented in the mockups is not yet finalized, there are some general design principles that have been followed in their creation. The most prominent stylistic aspect is the relatively dark overall look of the user interface, coupled with the conservative use of colors.

Since most content accessed with personal computers is bright in nature (websites, documents, and other textual content is usually presented on a light background), the contrasting darkness of the interface aids in lifting the actual content up into the focus of the user. While it might at first seem paradoxical to put a considerable amount of effort into designing an interface that ends up trying to stay out of the spotlight, ultimately the content — not the user interface itself — is the most important element on screen for the user. A dark interface also saves electricity by requiring less power from the display.

With minimal use of color, that property can be effectively used as a highlighting method; with less interference from other colored elements, a batch of color immediately stands out from a predominantly grayscale environment. The dark interface also further helps in bringing out the intenseness of any colored elements.

The use of a dark, relatively colorless interface is not without its problems, however. Since *Stripes* is designed to be an accessible, friendly environment, the dark visual look-and-feel might in fact trigger the opposite reaction in first-time users. Also, dark user interfaces tend to emphasize reflections on screens made out of glass or other highly reflective materials.

3.1.2 Resolution-independence

With the arrival of handheld personal computers, small-scale screens have become increasingly frequent. While the physical size of these screens is restricted by human factors, such us the size of our hands and pockets, and sharpness of our eyesight, the resolution of small screens is progressively increasing; this leads to larger pixel densities, or the amount of pixels per inch (PPI), and the densest small screens are already nearing PPIs of 600. As a comparison, desktop screens that have been increasing in diameter as well as resolution have retained a PPI or approximately 100. Because of the difference in PPI values, graphical elements that are physically large on desktop screens tend to look much smaller on handheld devices.[41]

This situation has raised a challenge in user interface design: how to create interface elements that retain their physical size on both screens with high and low PPI values. One proposed solution is resolution-independence — providing the ability to scale user interface elements in terms of resolution.

41 At the same time, touch screens require relatively large user interface elements due to the fact that they are manipulated with the tip of the finger, not a much smaller and more precise cursor.

There are two approaches, or imaging models, to drawing graphics on screens — raster bitmaps and vectors; both methods have their unique characteristics, and present specific advantages and disadvantages over each other [Preece, et al., 1994, 290]:

- Most interfaces today are created with raster images, bitmapped graphics that are created pixel by pixel. Because of their nature, raster images do not scale well, especially upwards, requiring interpolation algorithms to fill in the missing information; up-scaled raster images tend to look blurred or fuzzy, and are therefore unattractive to use in user interfaces. Since scaling raster images down yields much better results, one approach is to create high-resolution interface elements for high-PPI screens, and simply scale them down when necessary.

- Resolution-independence could also be achieved by using vector graphics instead of raster images — a mathematical approach to creating graphics. Vector graphics are already widely used in print media, where graphics created with relatively low-PPI screens need to scale up to look good on printed paper, a medium with a typical DPI (dots-per-inch) of 300 or more. Vector graphics, due to their mathematical heritage, can be scaled up or down easily without the need for destructive interpolation methods[42], but have traditionally been considered more processor-intensive to handle than raster images, especially when more details are introduced to the graphics [Foley, et al., 1994, 10]. From the point of view of user interface elements, however, simplistic shapes and graphics are often desired, making vector graphics an interesting contender to raster images in terms of providing effective resolution-independence.

Since *Stripes* is designed to be usable on various screens with numerous resolutions and PPIs, and with touch and cursor-based input alike, all interface elements are designed with vector graphics and resolution-independence in mind. Whether or not this approach is too consuming with regard to processing power would certainly require extensive technical testing.

3.1.3 Concept of popularity

The concept of popularity is an overreaching design solution that tries to remedy the problems induced by feature bloat, customization, and overwhelming choice by sorting or highlighting elements based on their relative popularity. The fundamental idea behind the

42 Vector graphics are often considered poor when being downsized to small resolutions. For example, vector-based icons can become undistinguishable at small sizes in comparison to raster icons specifically created pixel by pixel.

concept of popularity is to reduce the amount of choice by prioritizing elements that have been proven to be important to the user by his or her prior behavior.

Popularity in itself is a well-known concept and easy to translate to user interface design: if a user clicks on an interface element called “A” ten times per hour and an element called “B” only once, element “A” is ten times more popular than element “B” (for that hour). Because of the dynamic nature of popularity, if the user then starts clicking element “B” more often while clicking element “A” less, the popularity of element “B” will start rising while the popularity of element “A” will start decreasing. In terms of user interface design, the most popular elements at any given time can be emphasized and thus become easier to locate and/or press. Emphasis based on popularity becomes increasingly beneficial the more elements are presented at a time. In a list of 100 files, a popular item is considerably easier to find if it is, for example, highlighted with a unique color, or increased in dimensions; According to Ware, there are several ways to make elements stand out from their surroundings, because of the preattentive processing of certain characteristics of visual stimuli in the human visual system [Ware, 2004, 149–154]. Due to Fitt’s law, an element that increases in size as it becomes more popular is not only easier to find, but also easier to hit (with a cursor-based input device).

The concept of popularity also advocates the general notion of the computer trying to learn from its user, not vice versa. Since the computer is actively tracking the user’s preferred interactions, it can slowly adapt its interface to match the needs of that particular user. Since popularity is relative and bound to time, a change in habit by the user’s part will be picked up by the computer and adapted to accordingly.

The concept of adaptive user interfaces has not been very popular in the industry, since not many efforts to use them have succeeded — Microsoft’s dynamic IntelliMenus being a prime example, as explained in subsection 2.4.2. Still, less obtrusive adaptive designs propose great potential benefits to end-users, and the concept should not be dismissed based on a few misfortunate efforts of the past. After all, the concept of adaptive interfaces can be realized in many ways. Paula Selvidge presents one case in web newsletter *Usability News*, produced by the Software Usability Research Laboratory at Wichita State University, where two different adaptive menu systems were tested against static menus:

> “Sears and [Shneiderman] (1992) investigated performance with different menu organizations in a word processing program: split menus, alphabetical menus, and frequency menus [as seen in Figure 25]. Split menus were created by splitting a menu into two sections, with a few of the most frequently selected items at the top of the menu and an alphabetical list of menu items below the frequently selected items. Alphabetical menus included alphabetically organized menu items, and frequency menus were reorganized by frequency

of item use. Thirty-eight participants completed 100 menu selections with each of the three menu types. The results indicated that split menus provided significantly faster access than both alphabetical and frequency menus, but no differences were found for error rates. Users preferred the split menus, followed by alphabetical and frequency menus. The authors concluded that split menus incorporated the ease of access for frequently selected items and the alphabetical ordering that is useful for low frequency items (Sears & [Shneiderman], 1992)." [Selvidge, 2002]

The danger of adaptive designs is two-fold: unpopular elements might eventually be hidden making them difficult to find when needed, and the position of elements might negate the benefits of spatial memory. These issues are considerable, and will be discussed with each occurrence of an adaptive design in *Stripes*.

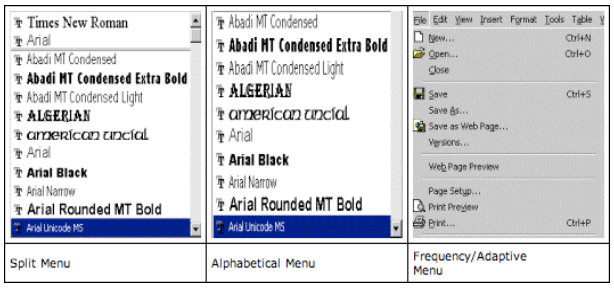


Figure 25 **From left to right: split menus, alphabetical menus, and frequency menus.**

3.1.4 Emphasis of text

Contrary to current trends of using many graphical elements in user interfaces, *Stripes* heavily utilizes textual elements and input, instead. This is an effort to combine the benefits of a GUI with those of a CLI.

Iconic representations are used conservatively when they provide considerable benefits. The reasoning is based on the problematic aspects of icons, namely the difficulty in their design for abstract[43] or culturally-dependent actions and commands, as well as the visual clutter they introduce; similar to the cautious use of color, a graphical element will stand out more if is not being surrounded by a multitude of other, nearly identical elements. With a large number of icons, they also tend to become indistinguishable from each other [Preece, et al., 1994, 95]. Although icons are generally used in an effort to increase the clarity of user interface elements, Raskin points out that the end-result is often contrary:

43 Just one example of a challenging metaphor is the save command. Even today, the old-fashioned, yet established icon of a floppy disk is widely used across software applications. While this representation might still resonate with older users, newcomers who have never seen a real floppy will be perplexed at best.

> "Instead of icons explaining, we have found that icons often require explanation. [...] The problem with icons can be considered an issue of diminished visibility: The interface presents an icon, but the meaning of the icon is not visible, or it may give the wrong message to someone for whom the graphic is unfamiliar or has a different interpretation." [Raskin, 2000, 168–169]

In *Stripes*, graphical representations are mainly used where their small use of screen real estate is necessary or when emphasizing an important element is desired.

In an effort to harness the power of CLIs, textual input in the form of search queries, commands, and batch processing is advocated. Often, users have a distinct idea of what they are looking for, in the form of language-based actions such as "save this document" or "check my email", and thus are better served by searching for that particular command than browsing. In essence, a user should be able to communicate with the computer just like with another person, using the most common form of communication, language[44], to do so. This is not to say that direct manipulation or using a mouse is less effective in all situations, but textual input provides a more effective — albeit also a more demanding — way to control and command a computer in many use cases. Despite advocating typing in general, the amount of characters that are required to be typed in should be minimized as much as possible, especially when considering the use of handheld mobile devices; providing a list of options after the first few characters have been entered, for example, allows for quickly accessing the desired item without the need of entering long strings of textual input.

The emphasis of text in interfaces is certainly a risky design choice, since the current trend in user interaction is direct manipulation and visually heavy interface design. CLIs feel old-fashioned, and the use of a keyboard requires more effort in comparison to the mouse. Fortunately, with the emergence of URLs and the search paradigm, keyboards are being used more and more in everyday computing, albeit in shorter bursts — accessing a website requires typing its address into a web browser (unless the site has already been bookmarked), and the common action of operating search engines, both online and offline, requires typing in short queries via the keyboard.

Moreover, iconic representations have become increasingly popular and widely used in user interfaces over the last decades: a general rule of usability dictates that humans are better at recognizing than memorizing [Nielsen, 1993, 129] [Preece et al., 1994, 118] [Raskin, 2000,

44 To be more exact, speech, not language per se, is the most common form of communication between humans (excluding facial or other bodily expressions). Speech recognition, however, is not yet at the stage of being a viable solution for mass-market computer interaction, nor is it the best solution in loud environments or for private interaction.

118], thus validating the use of icons and graphics over textual representation that often requires additional cognitive processing. Just like with all comprehension, the human visual system requires time and effort to parse through visual stimuli; the more visual elements an interface holds, the more difficult and time-consuming it is to identify and, consequently, manipulate them. As Ware notes, "the eyeball [is] an information-gathering searchlight, sweeping the visual world under the guidance of the cognitive centers that control our attention" whereby "[we] can parse somewhere between four and twelve items before the eye jumps to another fixation." [Ware, 2004, 145] Another issue with textual information is its arbitrary nature: as Ware points out, "the written word *dog* bears no perceptual relationship to any actual animal [Ware, 2004, 10]", whereas a graphical representation (such as an icon) is a sensory language processed much faster by the human visual system, not necessarily requiring any learning [Ware, 2004, 14]. Of course, a graphical representation is often a combination of both arbitrary and sensory languages, but their graphical nature itself can nevertheless help in their distinction and visual processing versus purely textual elements such as words.

Since textual representation over graphical representation has — like most user interface design decisions — both advantages and disadvantages, the challenge is to strike a balance between the two approaches and use each at appropriate times where their respective strengths outweigh their weaknesses.

3.1.5 Layered design

Roughly categorized, *Stripes* contains three layers each corresponding to a fundamental task of using a user interface: launching processes, interacting with them, and navigating between them. The division between these layers is not necessarily strict or even very visible to the user per se, but their underlying separation helps in creating a logical and simple interaction model.

In most other user interfaces, all these three tasks have been tied together so that, at any given time, the user can perform any task. This design approach results in highly compromised solutions that need to dedicate screen real estate and the users' attention between all three tasks, while often requiring several user interface elements to properly satisfy them. In Microsoft Windows, for example, launching processes can be done in multiple ways, via the Desktop, the Start-menu, and the Taskbar; none of these solutions individually creates a complete solution for launching processes, since the Desktop gets clogged up quickly and can be hidden by application windows, the Start-menu is cramped yet filled with a host of clickable elements, and the Taskbar can only hold a limited number of miniscule targets. Together, these three solutions compensate for each other's weaknesses, but from the users' point of view, there is no single, comprehensive way to accomplish the task of

launching processes: every time they wish to do so, they have to make a conscious decision between three mediocre options.[45]

Although the separation of the layers does require switching between them, the benefits are considerable. Dedicated screen real estate alone allows for far more comprehensive designs, while simultaneously decreasing the amount of required space for interface elements across all layers. A logical and clearly divided approach makes it easier for users to accomplish said tasks, since there are no competing solutions that all solve merely a part of the puzzle.

3.2 Boot screen

Albeit not by any means a highly important part of the operating system user experience, the boot screen in *Stripes*, as seen in Figure 26, has been designed to display at least some useful information, namely the current time and date, along with a customizable, full-screen background image similar to a Desktop wallpaper. The time it takes to boot up the operating system is calculated based on the duration of previous boot sessions, so a progress bar (or even a timer) can be presented with acceptable accuracy. Ideally, having

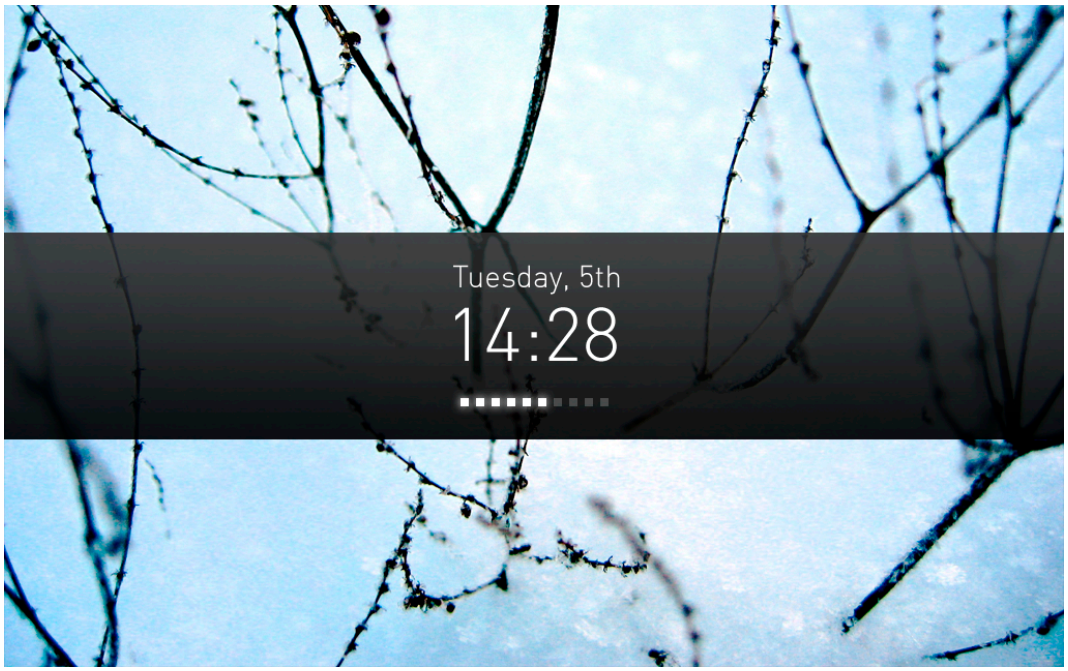


Figure 26 The boot screen.

45 Four, if search is included. In recent years, the search paradigm has become a formidable option for launching purposes, but requires typing and memorizing file names. More information on the search paradigm is presented in subsection 2.4.5.

the operating system boot up so quickly that the boot screen is never even displayed would certainly be the best possible design solution; in this case, the boot screen could simply take the role of a login screen.

3.3 Common UI frame

Although the design approach in *Stripes* is heavily based on the separation of user interaction into several layers as described in subsection 3.1.5, a common user interface frame is provided for accessing these layers as well as system-wide controls and status information.

As seen in Figure 27, the common UI frame is comprised of two bars — one at the top and another at the bottom of the screen:

Figure 27 The common UI frame.

- The top bar contains three elements, a power switch on the extreme left, the current date and time in the center, and status indicators on the extreme right. The power switch allows for shutting down or restarting the system, as well as entering other power-related states such as a standby mode. The date and time indicator is centrally located to underline its importance. Various system-wide status indicators, such as the status of the wireless network connection and the current volume level, are presented with clear, simplified icons that extend into contextual menus when selected.

- The bottom bar presents two buttons, both at the extreme ends of the bar. These buttons are used to navigate between the launching and navigation layers of Stripes. Because of their importance, they are bigger in size and contain distinct icons. Also, both buttons have a small background area for highlighting purposes. The position of the bar is designed to help with touch input: pressing these commonly used buttons does not obscure the screen with the hands of the user.

Since all interactive elements in the common UI frame are located at the corners of the screen, they are easily selectable with a cursor-based input device such as a mouse according to Fitt's law; since the cursor will stop at the edges of the screen no matter how fast or far the input device is moved, the size of any screen edges and especially corners is virtually unlimited (in terms of the law).

Because process launching, interaction, and navigation have been separated into their respective layers, the common UI frame can remain small in size and retain simplicity by holding a bare minimum of user interface elements. This design approach, however, does come at the cost of requiring an extra click per action when compared to a Dock or the Taskbar's Quick Launch bar (both described in subsection 2.4.1), as launching a process or switching between application windows with the common UI frame necessitates two clicks instead of one. In terms of navigation, there is a limited shortcut method discussed in more detail in subsection 3.7 that tries to minimize this drawback.

3.4 Launch screen

After booting up the operating system, the user enters what could be considered the starting point of interaction, the launch screen. The launch screen corresponds to the process-launching layer of the user interface, and, as seen in Figure 28, the left button in the bottom bar of the common UI frame is highlighted respectively. Whenever a user wishes to launch new processes, the launch screen provides a dedicated, highly optimized place to do so.

The launch screen is designed to allow for quick access to popular applications, files, and websites, as well as presenting useful, relevant information at a glance. The screen space is divided into three sections, the light-colored top bar dominated by a large input text field, a fairly large list of applications, files, and websites, and a strip containing applets — miniature applications with limited functionality that mainly display information provided by web services or local applications. Each section is explained in further detail below.

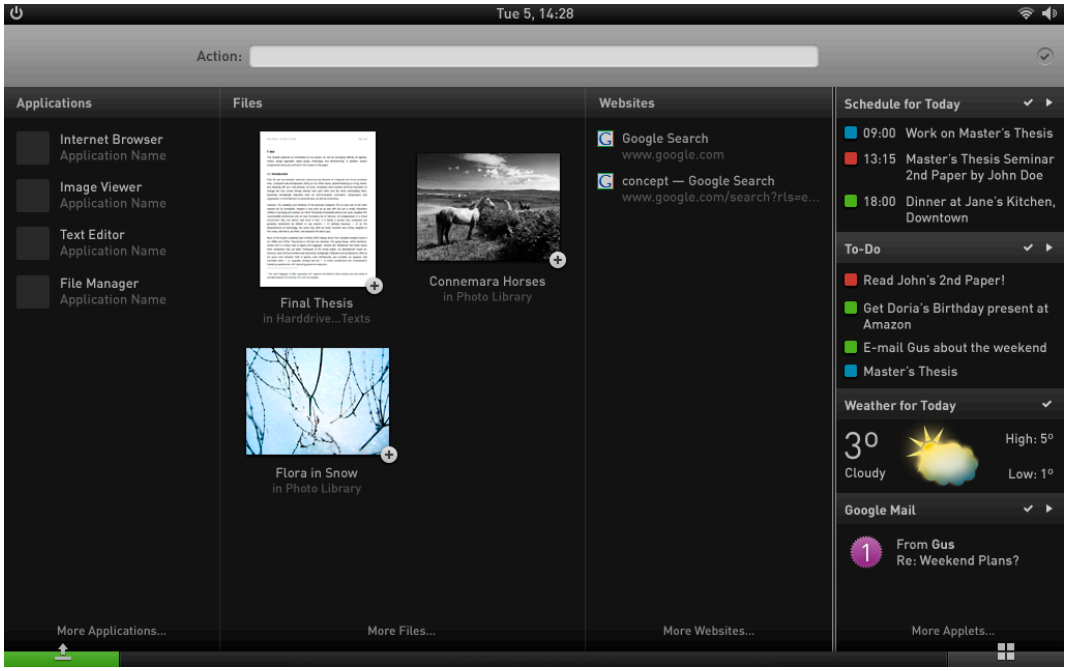


Figure 28 **The Launcher screen is automatically presented after booting up the operating system.**

3.4.1 Action bar

The top bar, also called the action bar, is presented in a contrasting color to the rest of the launch screen in order to highlight its importance. The assumption is that whenever users enter the launch screen, they want to start a new process, access a particular file, open a specific website, or, generally speaking, perform an action of some kind. This is particularly true when booting up the operating system — after all, there always is a very specific reason to do so; a user might think to herself: "I want to check my email", and this desire leads into booting up the machine. With a clear agenda on her mind, the fastest way to perform said action would be to tell the computer to "check my email" as soon as the machine is operational. The action bar provides this opportunity by allowing for typing in the desired action.[46]

The action bar, in essence, allows for a dialog between the user and the computer, and is highly flexible: it understands search queries, web URLs, application functionalities, and calculations. Being text-based, it is easily extendable to nearly any type of process launching. Using metadata to tag application functionalities, and with the help of an inte-

46 There is certainly no reason why spoken commands couldn't be integrated into the user interface once speech recognition becomes reliable enough.

grated thesaurus and dictionary database, advanced operations can be performed easily and with reasonable reliability; when issuing the command of checking email, for example, the action bar should be able to understand variations of the action, such as “get email” and “check for new mail”. Since the action bar performs just like a search field, all results are presented in a drop-down list once a query has been entered into the bar, making it possible to choose the correct action even if automation fails.

Because of the immerse flexibility of the action bar, learning how to use it properly might be an overwhelmingly daunting task for inexperienced and new users. In order to show how the bar can be operated, all actions that the user performs at the launch screen while *not* using the action bar display the respective command query inside the action bar; hovering the mouse cursor over a web browser application, for example, would show the words “open web browser” or “browse the web” in the action bar.

Similar to speech recognition, the success of the action bar is highly dependant on how seldom errors occur, as well as how well its flexibility and power is presented to users; building up enough trust in users’ minds requires fail-proof operation and some initiative to try out the bar in the first place. If successful, however, the action bar becomes a highly productive way to accomplish tasks in the most natural way humans communicate — language. Sending an email to a known friend, for example, would only require one step instead of ten:

1. typing in “send email to John” into the action bar, and pressing enter

versus

1. remembering the email application’s name,
2. locating its icon or name from a list of choices (possibly requiring opening one or several menus),
3. moving the cursor/finger over the icon,
4. activating that icon (possibly requiring two clicks),
5. locating the button to compose a new email message,
6. moving the cursor/finger over the button,
7. activating the button to compose a new email message,
8. locating the recipient field,

9. moving the cursor/finger over the field, and

10. typing in “John” in the recipient field.

While many of these actions might be quick to perform, and can be accomplished using keyboard shortcuts, the list serves merely to highlight the amount of reduced steps when using a command line interface coupled with intelligent search functionality instead of a traditional GUI approach. The actual performance of accomplishing the task is of course dependant on how well and quickly a user can type versus using a mouse (or a touch-based interface), but the action bar nevertheless reduces cognitive load and eliminates several visual search operations.

3.4.2 Popular items

The largest section of the launch screen is dedicated to three columns containing popular applications, files, and web addresses. This section can be considered a Desktop replacement, providing access to important processes with a single click.

As described in subsection 2.2.1, one of the most critical issues regarding the Desktop is its inability to provide easy access to important processes due to the constant management it requires. Important processes are, in most cases, defined by the amount of time users spend with them, and how often they access them; a file that has been under constant work for the last week is certainly more important to the user at the end of the week than many others that have not been touched in months. Utilizing the concept of popularity, processes can thus be ranked by their importance. Since there are far more applications, files, and visited websites located on local storage than could possibly be displayed in a single list, the launch screen displays only the most relevant items instead, negating any need for manual management of the lists; it is highly likely, after all, that users only have a handful of items that they are interested in at any given time — all items not ranked high enough in popularity to make it into the lists can easily be accessed via the “more”-buttons at the bottom of each list or the action bar, instead. In addition, options for “pinning down” certain items and removing them from the lists are provided for added control.

Because of the adaptive nature of the lists, items are shuffled around on a regular basis. This proposes a problem for spatial memory, which cannot anymore be utilized to find items in a rapid fashion. In order to counter this problem to some extent, all applications are

presented with their respective application icons, files with their preview thumbnails, and websites with their favicons[47] or actual page thumbnails, which should help in finding items based on their appearance, not location. Similar adaptive popularity lists have already been widely used in the form of "recent file"-lists, as well as in Microsoft Windows' Start-menu, as described in subsection 2.4.2 — solutions that have, unlike IntelliMenus, stayed in the recent versions of the operating system (and therefore at least partly proving their general acceptance).

Replacing the Desktop with lists of popular items remedies all of its biggest problems: file management is reduced, relevant and unfinished files are readily available, screen real estate is efficiently used, the launch screen serves as a true starting point for the user, and the old, aging metaphor is left behind. The much-appreciated Desktop wallpaper is not completely removed, since it is still displayed on the boot screen.

3.4.3 Applet strip

The applet strip, located at the right-hand side of the launch screen, contains miniature helper applications that aim to bring relevant information to the forefront without the need of launching dedicated applications — much like Widgets discussed in subsection 2.4.9. The applets can also be web-aware, and pull information from online services.

Applets can be manually added and removed from the strip by using the "more applets"-button at the bottom of the strip, while the strip itself is resizable and can contain multiple columns of applets. Individual applets can be configured (if needed) via the check mark-icon located next to the header of each applet, and a link is provided for launching the parent application or web service.

Because applets are only shown at the launch screen, they do not reduce the amount of screen space when using applications (as is the case with the Windows Sidebar in Windows Vista), yet still are clearly located within the visible user interface (as is not the case with the Dashboard in Mac OS X). This design approach should reinforce their presence while not compromising on screen space; while applets are not visible at all times, they are easily accessible by entering the launch screen.

47 Due to the increasing popularity of the Apple iPhone, many websites now contain a larger favicon specifically designed to be used with that particular device. A similar approach, or even the exact same graphic, could be used for website identification on the launch screen.

3.5 Windowing model

Once a process is launched, be it an application, a file, or a website, the user enters the application interaction layer of the user interface containing all application windows. Unlike in traditional GUIs, where all windows are stacked on top of each other, in *Stripes*, windows are placed next to each other in a row — or strip. By default, only one application window is in focus at any given time, at maximum size, without any overlapping from other windows, as shown in Figure 29.

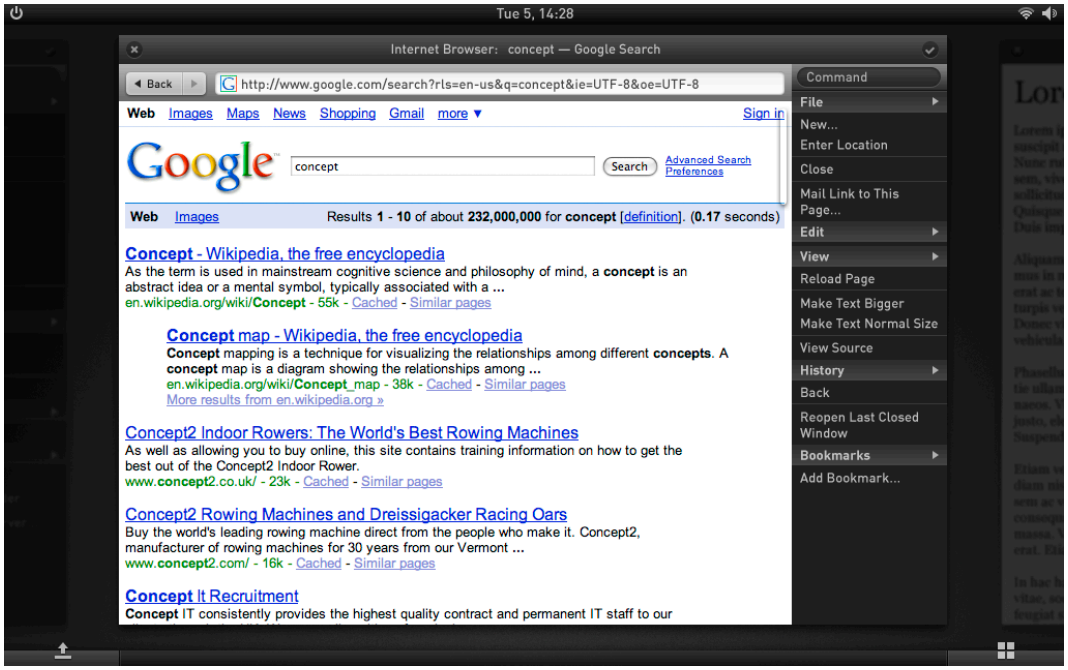


Figure 29 **Only a single window is in focus and completely visible by default. The next and previous windows in the strip are partially visible.**

The reasoning behind the single-window approach is to give the user complete cognitive and visual focus on the task at hand. As pointed out in subsection 2.2.8, humans are more productive when we are able to dedicate our attention to a single task (unless the task is trivial or habitual) than if we are presented with a multitude of simultaneous tasks. At the same time, the single-window approach removes visual clutter and attention-breaking distractions produced by other application windows. Because nearly all of the screen space is reserved for a single application window at any given time, the window can be of maximum size and does not need to be resized; because a window can never occlude other windows, there is no need for it to be moved out of the way, minimized, or hidden — removing the need for unnecessary window management. Furthermore, the single-window approach effectively removes any window focus problems, as described in subsection 2.2.8, where users accidentally misread an inactive window as the active one; furthermore, new windows open up in consistent, expected locations on the screen, namely in the center,

requiring no repositioning whatsoever. Lessened functionality also reduces the amount of user interface controls, diminishing both visual and cognitive load even further.

Despite the single-window approach, there is a way to place several windows on one screen for comparison purposes, as discussed in more detail in subsection 3.8, but this functionality is reserved for special cases, only. Also, windows can be freely rearranged in the strip, as well as grouped into task-specific rows. With effective navigation methods, presented in more detail in subsection 3.7, users can quickly switch between windows and get a complete overview of the workspace.

In an effort to unify and simplify towards a windowing concept, tabs are not used in association with application windows per se. The main benefit of tabs, as described in subsection 2.4.6, is the reduced amount of application windows, and the grouping of associated windows under a parent window. In *Stripes*, both benefits are gained without the use of tabs. In many ways, *Stripes'* windowing model combines the grouping options and simplistic management of tabs with the traditional model of application windows, where windows are easy to recognize and can be freely organized.

Whenever an application is launched solitarily (as opposed to opening a file which also launches the accompanying application), and if said application utilizes a single document interface design (as opposed to a single-window application[48]), the users are presented with an application-specific start screen vaguely similar to the launch screen discussed in subsection 3.4. The start screen allows for creating new documents, quickly accessing previously opened documents, and may contain application-specific launch options — such as accessing bookmarks. Figure 30 presents the start screen for a web browser.

Start screens, or welcome screens as they are sometimes called, are not a new innovation by any means. In *Stripes*, however, start screens also utilize the concept of popularity to display items that are most likely to be of interest to the user (albeit a list of recently opened files negates the need for this functionality in many cases). The fundamental idea is simply to provide easy and quick access to relevant files. Start screens can also be enhanced with search and filtering capabilities, allowing for reducing the amount of items in the lists; in web browsers, entering a partial URL into the location field can similarly filter out mismatching items.

48 Single-window applications can be defined by only containing one main window. They do not contain tabs, multiple sub-windows, or even separate document windows — just a single application window.

Once an item has been selected from the start screen, it is replaced by the newly opened window. The start screen can be accessed at any time via the menu strip and a dedicated user interface element within the application window. Essentially, an application may not be running without having some kind of a visual presence, be it one or several document windows or, if no windows are open, the start screen.

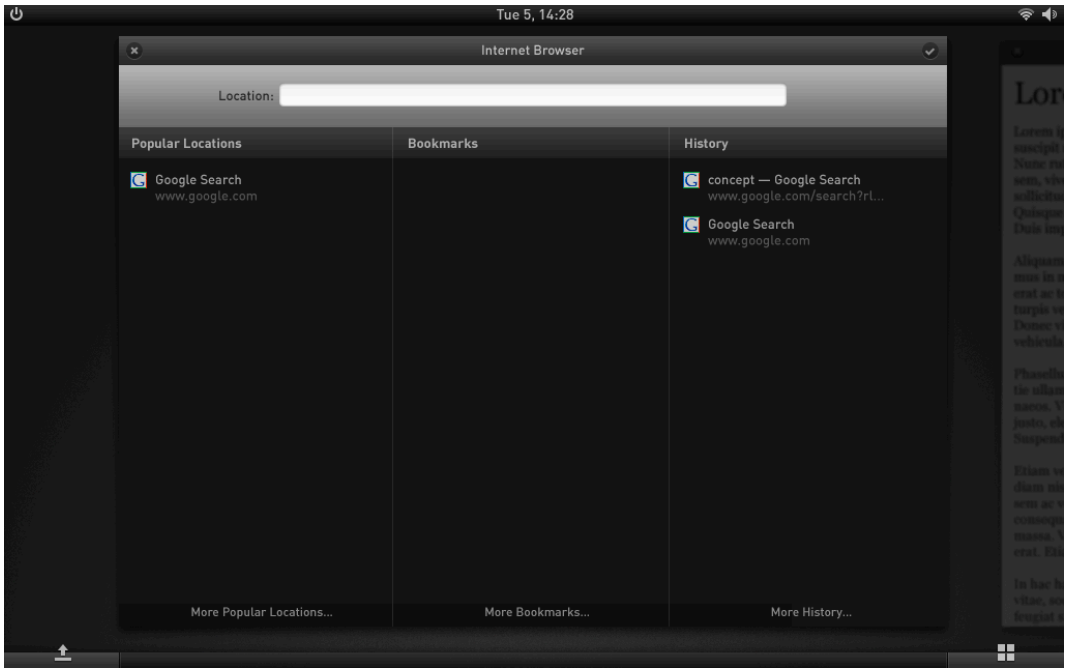


Figure 30 The start screen for an internet browser application.

The change in the windowing model of *Stripes* is perhaps the most radical departure from the traditional model of stacking application windows on top of each other. As such, it would probably cause initial confusion with users used to the norm. Fortunately, many operating systems found on mobile handheld devices have adapted to the single-window model due to limited screen space, and their growing popularity in part validates this atypical windowing model. The benefits of reduced window management, simple navigation, increased productivity, and general ease-of-use should eventually outweigh the unavoidable learning curve.

3.6 Issuing commands within applications

One of the increasingly growing concerns in operating system (and application) user interface design, as described in subsections 2.2.2, 2.2.3, and especially 2.2.5, is the way commands are presented and issued; the amount of commands increases with the amount of features, which in turn increase due to marketing pressure. In the past, there have been several attempts to reduce the visible amount of commands in various ways, one attempt

— Microsoft's IntelliMenus — being one of the most infamous examples, as discussed in subsection 2.4.2.

With the appearance of handheld mobile devices that are limited in physical screen size, there is a desire for issuing commands without resorting to a classic menu-based command bar. This desire has led to further innovation, such as gesture-based commands, as described in subsection 2.4.4, and the use of context-sensitive command elements; the notion of user interface elements appearing and disappearing based on the item in focus allows for interacting with the context in a more direct, uncluttered manner. Despite the benefits of this approach, however, there are drawbacks as well: user interface elements are generally hidden out of sight, (making it more difficult for the user to grasp what actions can and cannot be executed), there is no consistent location where commands are found (making it more difficult to develop a habit of locating commands quickly, partly due to reducing the benefits of using spatial memory), and context-*independent* commands still pose the same problems as before. Especially gestures quickly become inadequate when more complicated tasks are encountered; while it is certainly intuitive and easy to browse photos with a finger gesture instead of pressing virtual buttons, it is considerably more difficult to start editing them without the introduction of traditional user interface elements.

Since *Stripes* is designed for a wide range of applications, ranging from simple to complex, a command system that is both flexible and powerful is desirable. The approach is to highlight relevant and context-sensitive commands while not compromising the ease of locating and executing any other commands — all accessed via a uniform, non-moving command palette. The approach is similar to the Ribbon interface, discussed in more detail in subsection 2.4.2, but with the added functionality of search and the concept of popularity along with other design deviations.

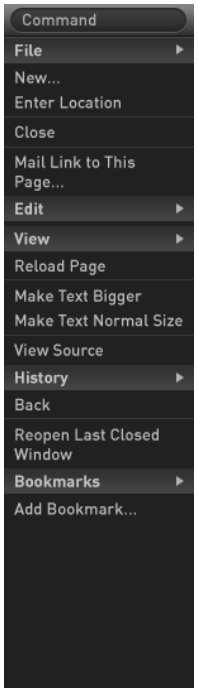


Figure 31 **The command strip of a web browser with several popular commands being presented.**

As shown in Figure 31, all application windows in *Stripes* contain a so-called command strip. The commands strip is vertically oriented, and located at the side of the window (on the right by default, but this setting is user-definable based on the handedness of the individual user). The commands strip contains a search

field for quickly searching desired commands, and a host of command groups similar to the top-level command groups found in traditional menu bars. In addition, the extra space is filled with as many popular commands as possible, presented underneath their respective groups.

In addition, as shown in Figure 32, when an element within the content section of a window is selected, all related, context-sensitive commands are added (if needed) to the command strip, and highlighted accordingly. If context-sensitive commands do not fit into the strip, other popular commands are momentarily removed from the list to ensure that *all* context-sensitive commands are visible when summoned; users are, after all, more likely to be interested in manipulating an element they have just selected than other, context-independent commands.

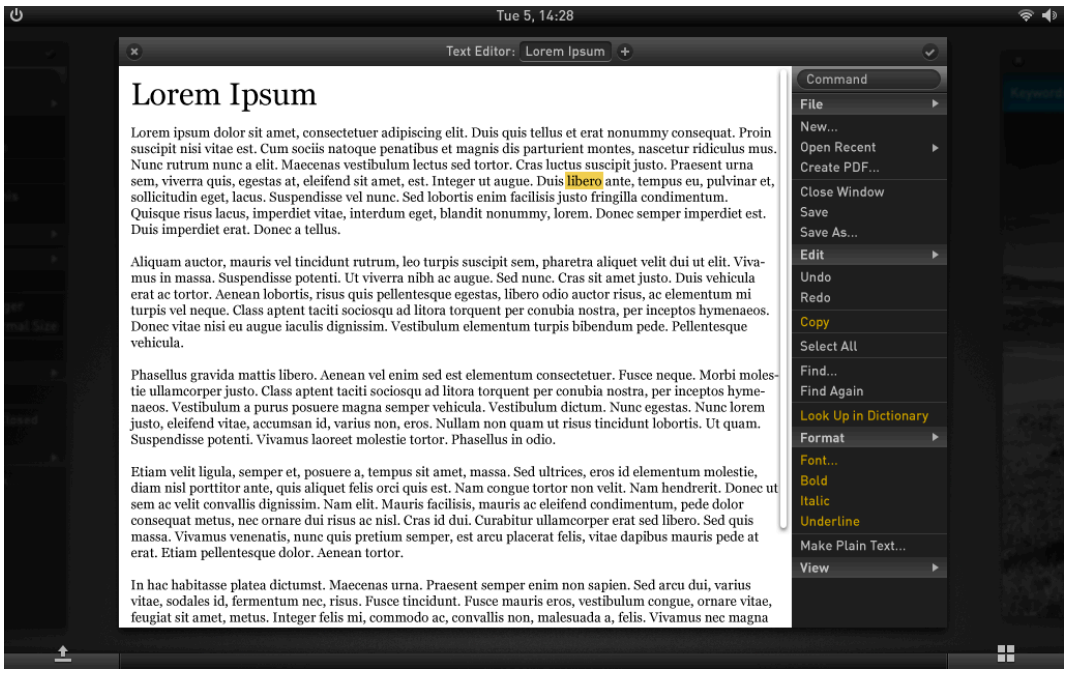


Figure 32 The command strip of a text editor presenting contextual commands in response to a text string being highlighted. The color of the highlight and the contextual commands are both yellow.

The first element in the strip, the search bar, mirrors the idea of quickly issuing typed, textual commands, as described in subsection 2.4.7. It allows for the entering of commands extremely quickly without resorting to the mouse, and corresponds directly to our language and trail of thought as humans; when users want to export a file, that particular command is on their mind — being able to simply type “export” into the search field will filter the entire list of commands a given application contains, and present only those commands

relevant to exporting. The approach of searching reduces the time needed to browse for the command in long lists of menus.[49]

As in traditional menu bars, all commands in the command strip are grouped into top-level categories. This categorization is necessary in order to keep the strip usable, since it is highly unlikely that all commands can fit onto the strip at all times. Selecting a command group will present all commands inside it, similarly to traditional menu bars, but instead of using drop-down menus, the contents of the groups replaces the prior contents of the strip in an animated fashion

Because most applications contain a reasonable amount of command groups, there is plenty of unused space in the command strip in its default state which can be used for presenting individual commands; even with a modest vertical resolution of 640 pixels, as presented in Figure 32, the command strip can hold approximately 25 items total, including commands groups, and using a comfortably-sized font size — providing space for as many as 20 commands. What commands are being presented is based on their popularity: using a particular command often will result in said command being provided alongside the groups, allowing for issuing popular commands with a single click or touch instead of browsing to it via the groups.

It is important to note that when an application has never been used before, no popular commands exist. The strip only contains the command groups, just like a traditional menu bars. This approach is crucial, since it forces users new to an application to browse the menu groups to understand their structure and contents. Only *after* understanding how to browse and search for commands is the concept of popularity introduced; Figure 33 presents the difference in

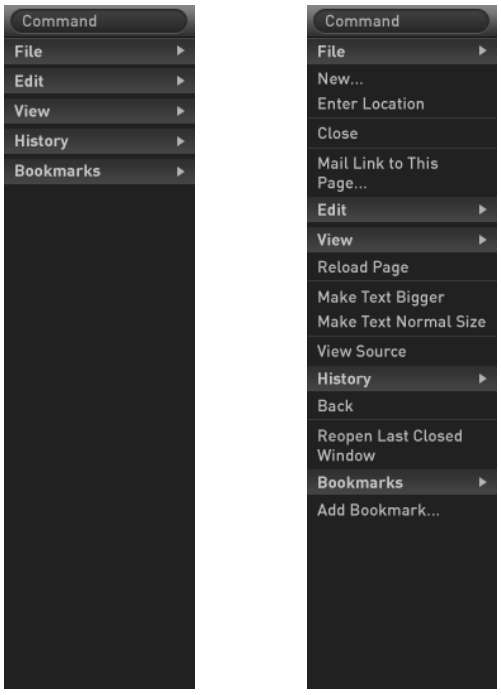


Figure 33 From left to right: The command strip of a web browser that has never been used before, and after prolonged use.

49 The fifth version of Mac OS X added a search field into the menu bar, under the help menu, which allows for quickly finding menu items. A similar idea is also being tested by the Office Lab team for the Ribbon interface [Microsoft Office Labs, 2008].

the command strip when an application has never been used versus after prolonged use. In addition, no commands are ever hidden by default, sidestepping the two main problems of IntelliMenus. Using the concept of popularity, applications that contain a large number of commands — of which usually only a limited set is important to an individual user — can remain usable and productive by dynamically adapting to the work habits of different users. Additionally, the burden of customization, as discussed in subsection 2.2.2, is reduced.

Because the command strip is context-sensitive, the need for separate contextual menus is greatly reduced. Due to the inherent visibility of the command strip, all contextual commands can now easily be seen, located, and issued without the need for trial-and-error. Also, the challenge of presenting contextual pop-up menus with touch-based user interfaces, as described in subsection 2.2.5, is eliminated. Finally, because contextual commands are shown in the command strip, they must also be found within the menu structure inside their respective menu groups — effectively removing the problem of allowing the placement of contextual commands solely in hidden pop-up menus.[50]

Keyboard shortcuts to commands are only presented when the dedicated command-key is held down, since touch-based devices might not include a hardware keyboard at all (and even if they did, shortcuts might be too challenging to execute given the small size of the keys). Naturally, a keyboard shortcut for highlighting the command search field is one of the most crucial shortcuts, allowing for entering commands in the middle of typing. Figure 34 illustrates how keyboard shortcuts are presented when the command-key is held down.

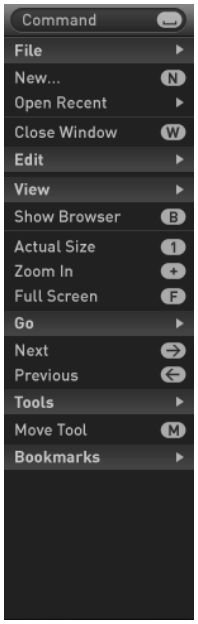


Figure 34 **The command strip of an image editor showing keyboard shortcuts when the command-key is held down.**

Despite the benefits of the command strip over traditional menu bars, the approach is certainly not without its flaws. Due to the vertical nature of the strip, the amount of screen space it occupies is fairly significant; the strip as presented in the mockups requires a

50 Separate contextual pop-up menus must not necessarily be removed completely; as long as they contain commands that are also accessible via the command strip, they can be provided as an additional feature to the system.

minimum of 150 pixels in width. Fortunately, with the increasing popularity of wide-screen displays, a vertical menu bar such as the command strip should be quite feasible. Another problem associated with the vertical nature of the bar is the handling of long command names, which have to occupy two or more lines, as well as the possibility of the strip not being able to contain all commands within a group, resulting in the necessity to scroll the list. Also, many commands that require additional input (changing fonts, entering URLs, selecting items from a drop-down list) and various tools in productivity software still require additional panels for holding those commands — partly ruining the concept of having all commands in one consistent location. In order to counter this problem somewhat, any additional panels that fit the horizontal layout should be placed next to the command strip, if possible. Furthermore, since the command strip doesn't utilize a drop-down menu design, browsing between the various menus and their submenus is considerably slower (ideally, the need for browsing menus should be reduced due to the emphasis on search). Whether or not these disadvantages outweigh the benefits of the command strip design needs to be determined by extensive user testing.

3.7 Navigation and getting an overview

Because of its single-window design, *Stripes* requires an effective way to navigate between opened windows. Navigation is accomplished in two ways, either by moving to the next or previous window in the application layer, or by zooming out to get a complete overview of the workspace. Both features are discussed in more detail below.

As seen in Figure 29, while one application window is in focus, the next and previous application windows in the row are partly visible. Selecting either one will move the focus to the respective window, effectively enabling single-click (or touch) navigation within the row. Animation is used in order to help users understand where they are moving.

Alternatively, users can zoom out of the workspace to get an overview of all opened windows and how they are ordered in the row, as well as directly zoom back into any window. The overview screen also allows for window management, discussed in more detail in subsection 3.8. As shown in Figure 35, the overview screen is activated by clicking on the overview-button located at the right of the bottom bar in the common UI frame, and is highlighted accordingly.

While the overview screen might seem identical to the Exposé functionality in Mac OS X and other operating systems, as described in subsection 2.4.3, there are some fundamental differences. For one, while Exposé rearranges visible windows into a seemingly random pattern, the overview screen truly zooms out of the focused window, providing a consistent, unarranged view of all opened windows. Furthermore, since application windows cannot be

hidden or minimized, the overview functionality provides a reliable and complete outlook on the workspace — a benefit that is further reinforced by the lack of tabs.

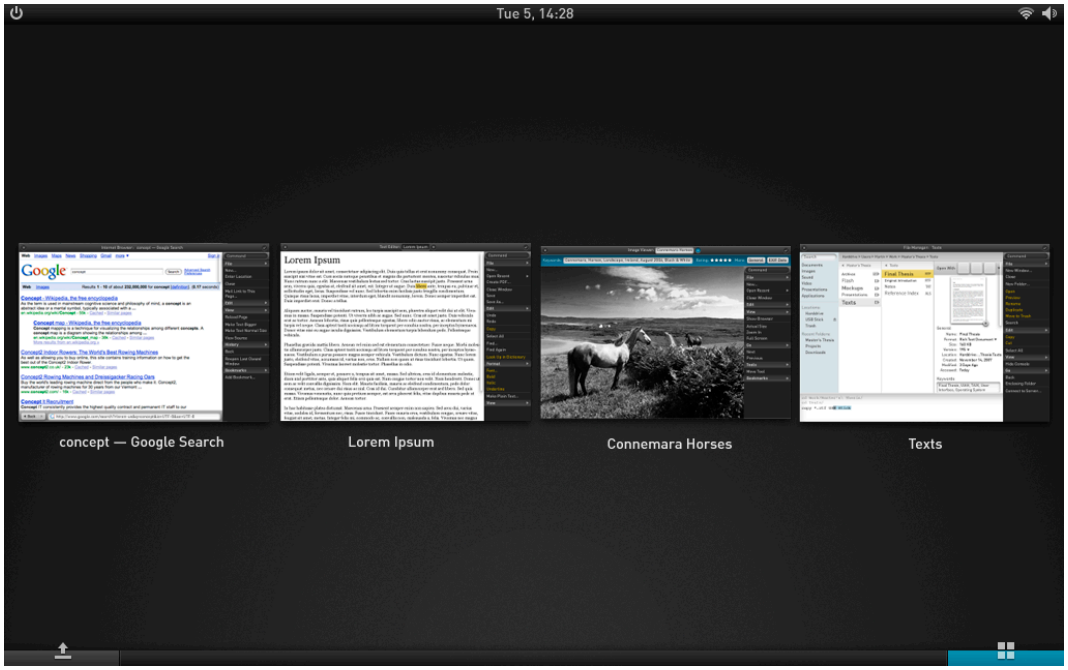


Figure 35 Zooming out of a window reveals an overview of the complete workspace.

One of the benefits of the traditional windows stack is the possibility for the user to arrange windows in such a manner that they can be accessed by a single click even if they are occluded by other windows (and are not completely hidden by fullscreen windows). While the navigation model in *Stripes* does not allow for such flexibility, it balances this drawback with simpler navigation that requires far less window management.

3.8 Window management

While one of the most prominent design approaches in *Stripes* is the reduction of window management, there are a handful of operations related to management that are simply too crucial to be omitted: rearranging windows within their respective window rows, grouping windows into task-oriented collections, and comparing two (or more) windows' content to each other.

All window management actions are performed at the overview screen. Windows are rearranged by simply dragging and dropping, similarly to how tabs are reorganized in modern web browsers. Windows snap into their new places, accompanied by subtle animation, for the sake of keeping window rows consistent in form. Grouping windows is performed similarly: windows cannot only be dragged horizontally, but also vertically above and below the

current window row in order to form new rows of windows. Each row therefore becomes a new collection — or group — of windows. Figure 36 illustrates how window grouping is performed.

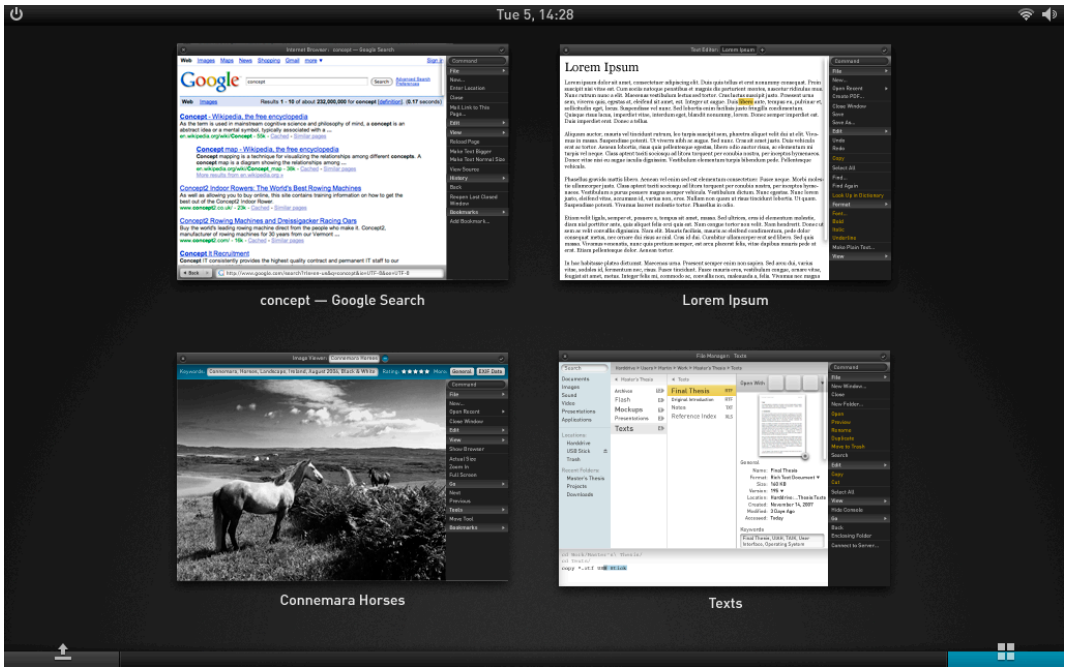


Figure 36 **Windows can be grouped by dragging them vertically outside of their current window strip. Here, two groups of two windows each have been formed.**

Although the windowing model in *Stripes* is based on a single-window approach, the ability to compare two documents, websites, or other items is a necessary feature. Two windows can therefore be temporarily merged together so that they form an application window pair. Windows are merged by dragging one on top of the other in the overview mode, and detached by dragging either one away. Figure 37 illustrates the merging action of two application windows.

Once merged, the window pair can be zoomed into just like any other window, as presented in Figure 38. By default, the size of each part of the window pair is equal, but their dimensions can be manually changed via dragging the separator line between them. Because only one of the windows can be in focus at any given time, the inactive window is visually de-emphasized.

Another design aspect related to window management is the treatment of panels. In current operating systems, panels are often considered as miniature, floating windows that usually contain user interface controls, but are not treated as actual windows by the system, as described in subsection 2.2.8. As with many contemporary applications, in *Stripes*, panels

are not floating, but rather part of the application window frame; this design approach fits the single-window nature of the user interface, and removes any occlusion problems associated with floating panels (at the cost of reduced flexibility).

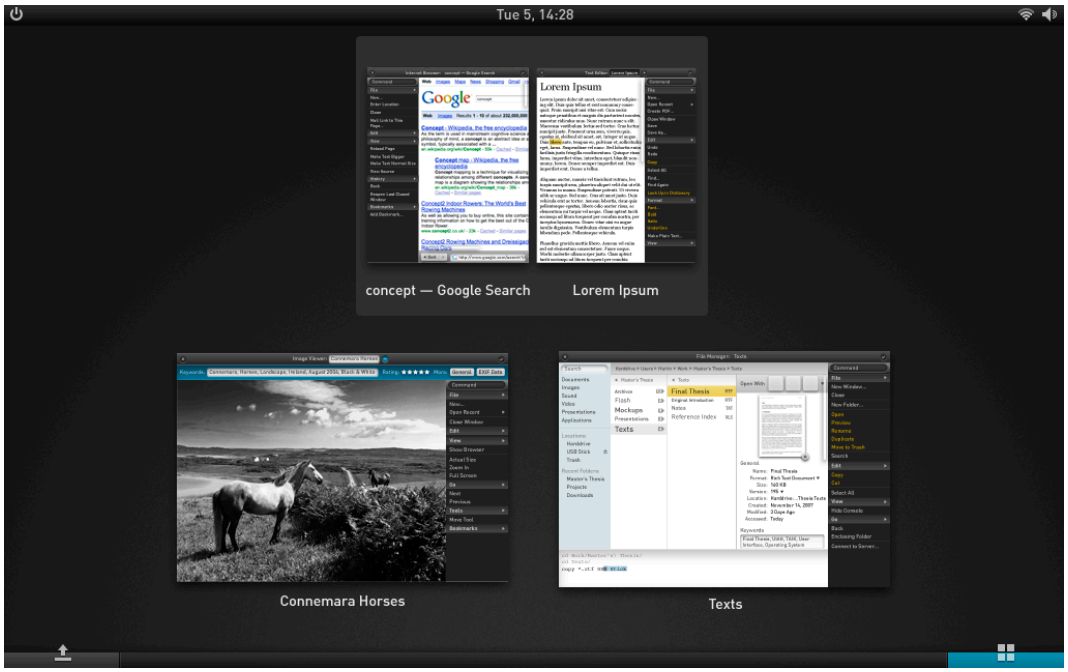


Figure 37 **Two windows can be temporarily merged to form window pair, allowing for comparing them side-by-side.**

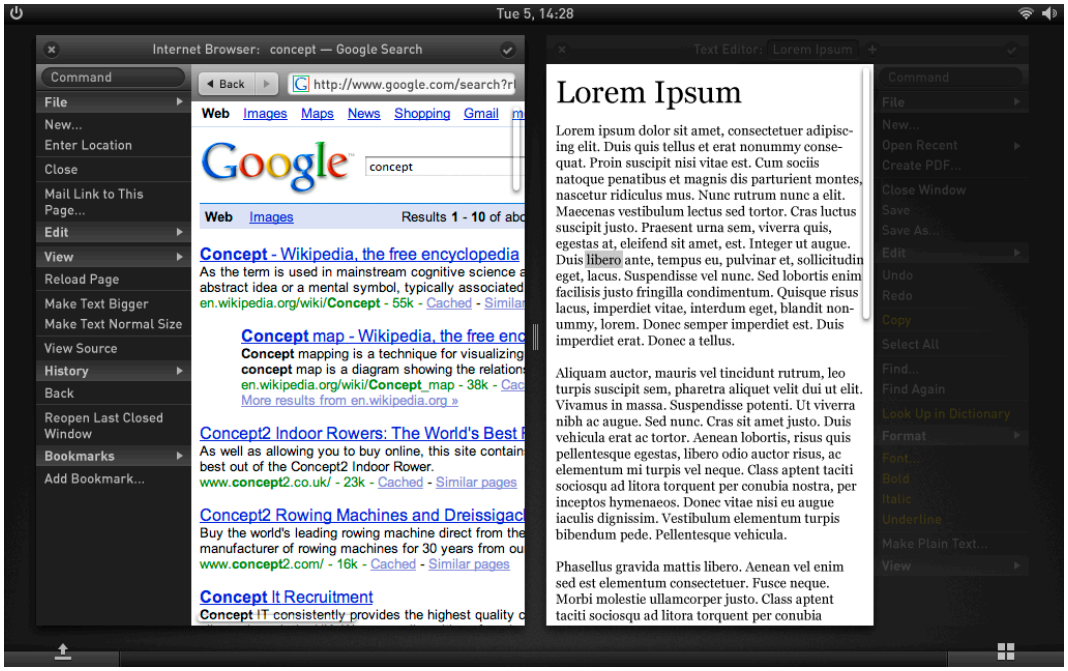


Figure 38 **Zooming into a window pair. The user interface controls of the unfocused window are visibly subdued.**

3.9 User interface details

This section provides insight into some of the user interface details in *Stripes* not related to the general design concepts and user interface solutions in prior sections. These additional features are presented in alphabetical order.

3.9.1 Accessibility

As mentioned in subsection 1.3, while *Stripes* is not specifically designed for people with disabilities or the elderly, for example, many of the design objectives and specific focus areas have been heavily influenced by the desire to improve general accessibility. Features such as the natural language-based textual commands of the action bar, and the integrated textual console (discussed in subsection 3.9.2) allow for improved accessibility for the blind, especially if coupled with speech recognition. The simplistic approach to window management, reliable overview of the complete workspace, the concept of popularity, and many other features should help non-technical users to locate and manipulate items with an increased sense of mastery, effectively removing much of the complexity of traditional operating system designs that make them inaccessible or incomprehensible to many. The system is designed with minimalist, clear, high-contrast, and scalable user interface elements that should allow for increased readability even for those with poorer vision, such as the elderly — and with complete support for touch and cursor-based input methods (as well as keyboards), the user interface can adapt to many different kinds of environments and use cases. Considering the collective amount of usability improvements combined with a more simplistic overall design, *Stripes* should be no less accessible than any other general-purpose operating system on the market, and in fact reinforce the notion of emphasizing accessibility in user interface design.

3.9.2 Console

In an effort to bring the power of command line interfaces to *Stripes*, the operating system integrates a console into all application windows in addition to offering a separate CLI terminal application as found on most modern operating systems. The integrated console provides quick and easy access to functions and commands associated with a given application, as well as any textual messages the application logs.

In an example seen in Figure 39, the user can easily copy all rich text format-files to an USB stick with the help of the console, a command that would be considerably more tedious to perform with the GUI alone. Prior commands performed using the GUI, namely drilling down in the file structure tree to the correct folder, are presented as logged messages in the

console. In addition, the console provides command completion functionality for added productivity.

If designed properly, the console should be able to perform all — or at least most — of the functionality the GUI provides, providing the user an alternative method to control the computer; if reliable, the console can be used in conjunction with the GUI to combine the benefits of both approaches when needed. As mentioned in subsection 2.4.7, there have already been efforts to bring back some of the power of command line interfaces to GUIs in various forms, and in this regard, an integrated console is simply an extension to the trend.

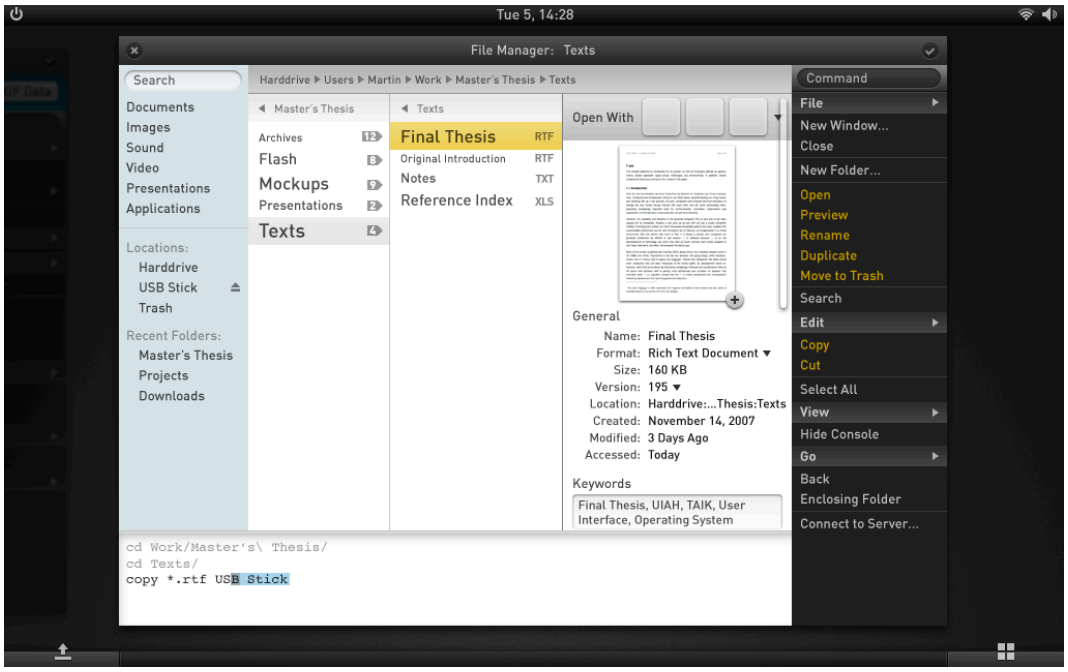


Figure 39 A file browser window with the integrated console in action.

3.9.3 Dialogs and notifications

Dialog and notification boxes are an often-seen user interface element in operating systems. Dialog boxes generally appear after issuing a complex command that requires additional user input or confirmation in order to be executed (such as converting a file from rich into plain text format), while notifications boxes simply provide informative feedback from the application to the user (such as affirming the completion of spell checking).

Traditionally, dialogs and notifications have been treated as pop-up windows that behave similarly as other application windows. As described in subsection 2.2.8, this design approach leads to the possibility of notification and dialog windows being hidden or

occluded by other windows, and — without a visual connection to the parent window — decoupling from the associated application window. The problem of decoupling has already been solved in Mac OS X with the concept of Sheets — sheet-like dialog boxes that slide out of the window's title bar and remain attached to the parent window until dismissed. While certainly an improvement over pop-up windows, Sheets tend to occlude the content underneath them.

In *Stripes*, both dialog and notification boxes are presented as a part of the application window, as seen in Figure 40. Dialog boxes behave much like Sheets, sliding out of the window title bar, but unlike Sheets in Mac OS X, they shift the contents of the window when presented. Also, the content of the window in question remains editable (if possible) even if the dialog is being presented. Notification boxes slide into view from the bottom of the application window, are displayed on a semi-transparent background, and are presented on top of the window content. Because of their design, notifications feel unobtrusive and do not shift content when being presented. They can also be easily dismissed by clicking or touching anywhere within the notification box area.

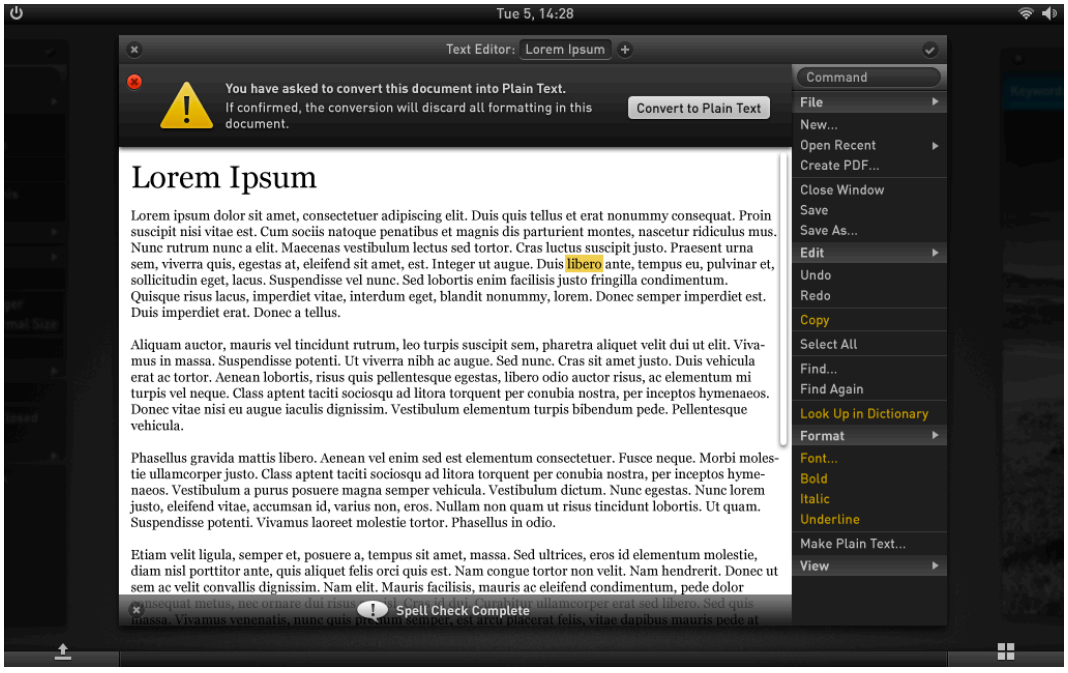


Figure 40 **Notification and dialog panels are attached to the main application window. Dialogs push the window content to avoid occlusion, notifications don't.**

3.9.4 File manager features

Despite industry efforts to make file structures and managers obsolete for the end-user, as described in subsection 2.2.4, the need to copy, move, delete, and otherwise manage files

directly within the file system persists as a common task. Optimally, in *Stripes*, users should not need to use the file manager very often, as relevant files are always readily available on the launch screen, as detailed in subsection 3.4; still, there are several improvements to the concept of a file manager that should further help the user when any management needs occur.

A mockup of the file manager is presented in Figure 39, which illustrates several design concepts. These features are explained in further detail below:

- The file manager contains a source bar, located at the far left of the application window. The source bar aids in filtering folder contents, and provides quick access to external media and any recently visited folders. The bar contains a search field for quickly filtering contents by name or other metadata, and further provides filters for narrowing down the search results by file type. The filters can naturally also be used when no search query has been entered.

- The actual folder structure is presented in a column layout similar to the column view in Mac OS X: folder contents are presented in list form within a column, and selecting a folder will display its contents in a subsequent column to the right — allowing for quickly browsing back and forth in the file system. The current location of the user within the file system is presented as a breadcrumb path at the very top of the column layout.

- Folders and files are highlighted by varying the size of their name according to their popularity. This visual aid helps in finding often-accessed items within long lists. Because items remain ordered alphabetically (or in any other sort order as requested by the user), items do not shuffle around within the list even when their popularity changes. There is also the possibility to provide a split list, where shortcuts to the most popular items are provided at the very top of the list.

- In addition to their name, folder and file items within the columns are presented with some additional information; folders show the number of items they contain, and files present their file type. While the design of the column layout dictates a minimalist approach, theoretically it would be possible to present even more metadata next to the item names.

- When a file is selected, the subsequent column presents information on that particular file. Most prominent is the “open with”-bar containing various applications associated with the given file type. This design allows for easily selecting the appropriate application for opening the file with, a problem discussed in subsection 2.2.4. In addition, a thumbnail preview of the file contents is presented, along with the option to

enlarge the preview to full screen. Most important metadata is presented below the thumbnail, along with user-defined keywords. In addition to metadata information, some additional features are also provided: file conversions can be initiated by simply selecting a different file format, and if the file system is backed up on regular basis, prior versions of the file can easily be accessed and retrieved.

- The file manager works well in conjunction with the integrated console, discussed in more detail in subsection 3.9.2, as batch operations can quickly be issued with the help of typed commands.

While Stripes does not remove the notion of file systems, or try to provide a database-like model for file management, several improvements to the file manager application — in addition to the launch screen — should minimize the need for tedious file management. Additional improvements to saving files are described in subsection 3.9.5.

3.9.5 Metadata

As outlined in subsection 2.4.5, metadata has become an increasingly important tool for providing additional information on files, allowing for more intelligent search results based not only on file names, but also on content. In *Stripes*, for example, all applications should provide information on their purpose, actions, and available commands that can be utilized in natural-language search queries, as detailed in subsection 3.4.

In addition to prominent metadata support in the file manager, as presented in subsection 3.9.4, application windows integrate metadata support for all windows that hold file-based content, such as documents, presentations, or audio files. Figure 41 shows an example of a photograph opened in an image viewer application, and how the file's metadata can be accessed without the need of the file manager application.

The top bar of an application window always contains the name of the application in question, and if it holds file-based content, the name of that particular file is displayed next to the application name inside an input field, followed by a small button for accessing file metadata. Selecting the input field allows for renaming the file on the fly[51], while clicking the button slides out a metadata bar containing all relevant, writable metadata informa-

51 Typically, file names cannot be changed without saving the file with a different name. If the file name (as seen by the user) is treated as metadata, and the actual file identifier (as used by the file system) is assigned by the computer, it should be possible to rename opened files on the fly, assuming the file identifier is renamed to reflect the file name as soon as the file is closed or saved.

tion associated with the file in question. In the example mockup, the photograph contains several user-definable keywords and a rating in addition to providing access to read-only, general metadata information as well as EXIF data provided by the digital camera that was used to take the photograph.

The emphasis on metadata promotes defining files in terms of descriptive information instead of just a file name, providing more reliable search results, and allowing the user to rename and characterize opened files at any given time. This avoids breaking the flow of work when saving files, as discussed in subsection 2.2.4. Ideally, the system should save all files automatically to a temporary location on the hard drive based on the file type (all documents are saved within the predefined documents-folder, and all video files within the video-folder, et cetera), enabling — not forcing — the user to name and place files with more precision when they see fit to do so.

Figure 41 Metadata support is integrated into all application windows.

3.9.6 Multi-screen support

Traditionally, multi-screen Desktop setups have been treated as an extended Desktop environment, where the workspace is stretched across all available screens. Alternatively, screens can mirror the contents of the main display. While extending the available workspace is certainly useful, due to the prevailing windowing model multi-screen setups require constant window management and increase cursor movement dramatically; accessing a static menu bar, as described in subsection 2.2.5, or any other user interface element (such

as a Dock or the Taskbar) located on the main screen requires constant cursor repositioning from screen to screen.

Stripes' multi-screen support differs from the traditional model, partly due to its single-window design, and opts for an approach often used in 3D modeling and animation software, where a single scene is viewed from multiple angles simultaneously with the help of multiple cameras. Similarly, each physical screen used with *Stripes* is treated as an individual camera, or viewport, to the commonly shared workspace. The viewports are independent from each other, and can overlap, allowing for mirroring contents. As seen in Figure 42, all viewports contain the common UI frame, allowing for accessing the full range of user interface controls from each screen independently, without the need to move the cursor from one screen to another.

The model of treating each screen as a new viewport of the workspace (instead of extending it) allows for more flexible and understandable multi-screen support, while lessening the need for window management and cursor movement. The model does have its drawbacks, however, since application windows can no longer be extended to cover multiple screens. Screens with different resolutions can also pose problems, since application windows must be sized individually according to each display.

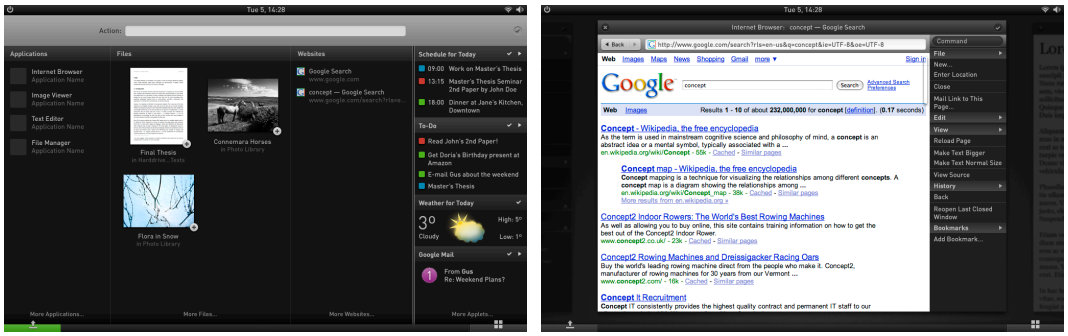


Figure 42 With multi-display setups, each screen is considered an individual view into the workspace.

3.9.7 Scrolling

As discussed in subsection 2.2.7, scrolling has become one of the most important user interactions when using a personal computer, mainly because of the Internet and the emergence of small-screen handheld devices. Despite its increasing importance, scrolling

itself has not become any easier or faster since the introduction of a hardware component, the scroll wheel on mice.[52]

Since *Stripes* is designed to be usable on touch and mice input alike, a unified scrolling mechanism that works fairly consistently with both is highly desired; users should be able to switch from one input device to another without having to adjust to a new scrolling system. The general design approach and features of *Stripes'* scrolling system are outlined below:

- As illustrated in Figure 43, the visual appearance of the scroll bars is highly subdued, presented as semi-transparent, simplistic capsules located on either the right-hand side or the bottom of the scrollable content (for vertical and horizontal scrolling, respectively). There are no scroll bar arrow buttons, and the bar itself cannot be grabbed, similarly to the iPhone scroll bars. The minimalist visual design allows for maximizing screen space for content.

- Due to the lack of direct scroll bar manipulation, scrolling is always performed with hand movement, regardless of whether the input device is a mouse, pen, or finger, and can be performed anywhere within the scrollable content area. Standard keyboard shortcuts for scrolling are also supported.

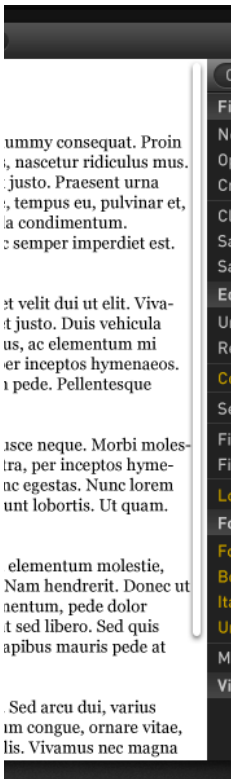


Figure 43 Scroll bars have a subdued visual appearance.

- One of the most prominent mismatches with gesture-based scrolling and many of the traditional scrolling methods on desktop computers is the direction of hand movement: with a finger swipe, scrolling is performed in the opposite direction of the swipe movement, while scroll bars, wheels, and modes (as described in subsection 2.2.7) scroll parallel to hand movement. To counter this disparity, scrolling in Stripes is

52 Certainly, scroll wheels themselves have evolved in the last decade, allowing for vertical and diagonal scrolling, as well as adding pressure-sensitivity.

always performed in the inverted direction of hand movement, unless a scroll wheel is used.

- Scrolling with the mouse is performed by clicking and holding down the secondary mouse button; due to the contextual command strip design, the need for separate contextual menus is decreased, and scrolling functionality is emphasized instead. Holding down the secondary mouse button will enter the user in a scrolling quasi-mode, where the direction of mouse movement is mapped to the inverted direction of scrolling, and the distance of mouse movement is mapped to the speed of scrolling. When the mouse button is released, scrolling ends. This approach removes the need for repeated dragging actions as witnessed when using a typical hand tool for scrolling purposes, and allows for quickly scrolling to the top or the bottom of the content.

- Scrolling with fingers (either directly or on a trackpad) is performed by touching with two or more fingers simultaneously and swiping. If the fingers approach the edges of the screen or trackpad, the scrolling quasi-mode described for mouse-based scrolling is entered, removing the need for repeated swiping gestures. Alternatively, swipes and dragging gestures can be treated differently, with the latter mapped to the scrolling quasi-mode.

- Scroll wheels on mice are supported, but not required. Unlike other scrolling methods, scroll wheels behave exactly as they do traditionally in all other operating systems. If necessary, separate contextual menus can be mapped to the scroll wheel, as it also functions as a tertiary mouse button.

While certainly different from the scrolling methods of any other operating system (both desktop and handheld variants), Stripes' approach to this fundamental interaction functionality is primarily designed to improve and unify the scrolling experience across all input devices. Despite the emphasis on faster, more accessible scrolling methods, however, the accompanying learning curve might prove to be too demanding for new users (especially if the scrolling methods are not specifically explained). Whether or not this indeed is the case requires thorough user testing.

4 Preliminary user test

This chapter describes the user testing that was conducted on an early demo version of *Stripes*. It provides an overview of the goals of the test, the testing procedure, the results, and any relevant background information.

4.1 Background information and technical details

In order to evaluate and validate most of the fundamental design aspects of *Stripes*, a round of preliminary user testing was organized with the overall goal of providing meaningful feedback on the design decisions and general acceptance of the user interface. At this relatively early stage of drafting out the look-and-feel of the user interface, as well as the feature set, it was vital to receive as many comments and responses as possible from potential end-users.

There are several types of user tests and approaches to testing. For this project, it felt necessary to built a working demo version of the user interface with a thought-out graphical style as well as enough functionality for a meaningful test; this approach allowed for a much more truthful and engaging testing experience than a simpler, paper-based prototype. The digital demo also had the advantage of being extremely mobile (the test could be brought anywhere with a portable computer), deployable online, and allowed screen capturing without the need of excessive camera and lighting setups. Additionally, building a digital prototype also allowed for better demonstration possibilities for presentations and lectures.

Moreover, working with conceptual designs is often challenging without actual prototypes, since many details of user interaction are easily forgotten or overlooked during the early design phases. Having a working product to look at and use, even if only on a rudimentary level, helps tremendously in fleshing out concepts and noticing any possible flaws in the design.

The *Stripes* demo was created in Adobe Flash due to its graphical capabilities. Using Flash allowed the prototype to include smooth animation, visual effects, and interactive elements with relatively little effort. On a personal level, the work that went into building the prototype as a dynamic, object-oriented, interactive product accounted for a major learning experience in scripting.

The prototype was designed for a resolution of 1024x640 pixels, a fairly small screen resolution, and contained most of the crucial features, albeit limited, described earlier in chapter 3. The following list is a detailed compilation of what was (and wasn't) included in the demo:

- The application started up at a simple boot screen with the date and time being displayed.

- The common UI frame included the date and time, a static power button, and no status indicators. Both the launch screen and the overview buttons worked as intended.

- The prototype included two demo applications, two files, and three non-interactive URL locations (Flash does not include a rendering engine for websites).

- The launch screen contained a partially working version of the action bar with simple keyword recognition, but no systematic search capabilities; the bar would recognize application and file names, as well as URL addresses, but would not filter and present query results. The applet strip was displayed, but was not interactive.

- Application windows could be created, closed, navigated, and rearranged, but not merged or grouped. Zooming out into overview mode was fully supported.

- Each application had a handful of working commands (all commands were, however, displayed). The command strip was fully functional, supporting popular commands, browsing of command groups, contextual commands, and search queries. The welcome screens were included and operational.

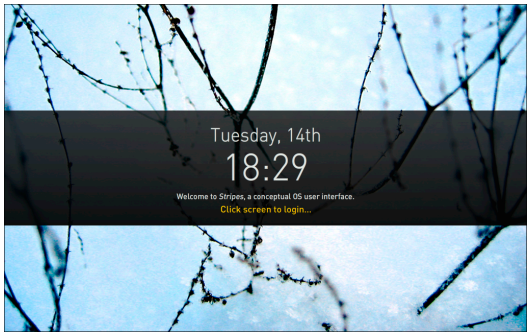
- Both notifications and dialogs were supported.

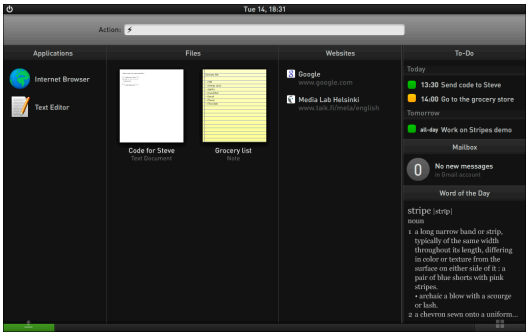
- Scrolling was not supported.

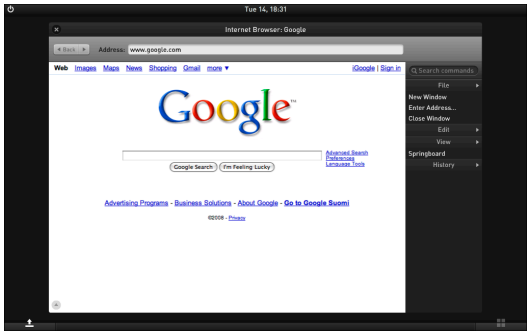
- File content was manipulatable, but none of the files had persistent states (they could not be saved) or supported metadata.

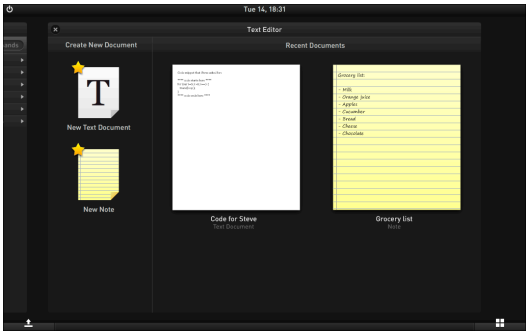
Figure 44 presents several screenshots of the prototype application that was used for the user test. The look-and-feel is not exactly identical to the original mockups, thus the prototype can be mostly considered as an iterative, subtly improved version to the original design; certain visual cues, such as the blur effect on unfocused windows, had to be omitted because they slowed down the prototype too heavily to warrant their inclusion.

Figure 44 **(Opposite page) Several screenshots from the Stripes prototype application.**

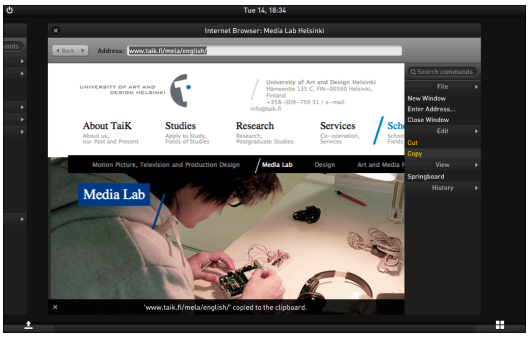

Tuesday, 14th
18:29
Welcome to Stripes, a conceptual OS user interface.
Click screen to login...

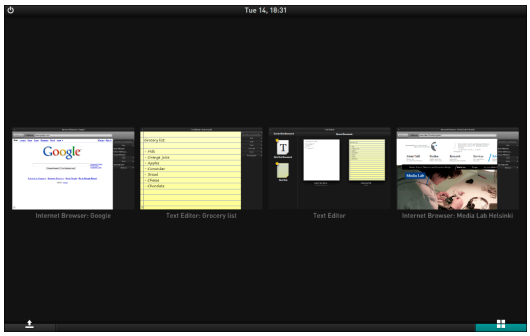

Tue 14, 18:31
Action:
Applications
Internet Browser
Text Editor
Files
Code for Steve
Grocery list
Websites
Google
Media Lab Helsinki
To-Do
Today
13:30 Send code to Steve
14:00 Go to the grocery store
Tomorrow
all-day Work on Stripes demo
Mailbox
No new messages
Word of the Day
stripe [strip]
noun

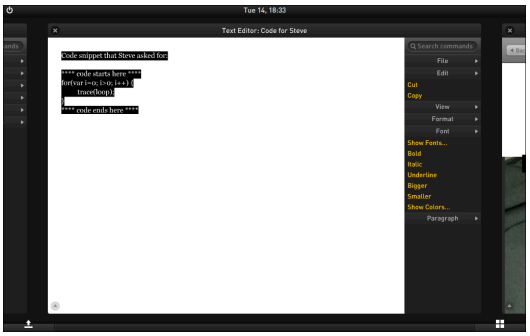

Tue 14, 18:31
Internet Browser: Google
Google
Google Search
I'm Feeling Lucky

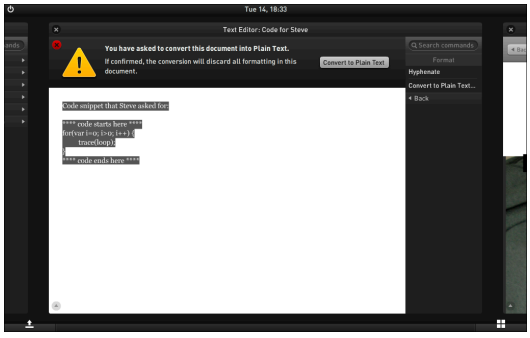

Tue 14, 18:31
Text Editor
Create New Document
Recent Documents
New Text Document
New Note
Code for Steve
Grocery list

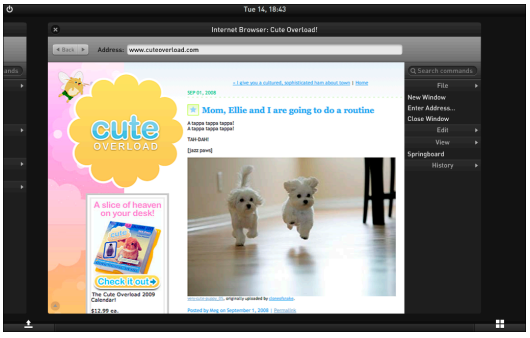

Tue 14, 18:31
Text Editor: Grocery list
Grocery list:
- Milk
- Orange juice
- Apples
- Cucumber
- Bread
- Cheese
- Chocolate


Tue 14, 18:34
Internet Browser: Media Lab Helsinki
About TaiK
Studies
Research
Services
Media Lab
'www.taik.fi/mela/english/' copied to the clipboard.


Tue 14, 18:31
Internet Browser: Google
Text Editor: Grocery list
Text Editor
Internet Browser: Media Lab Helsinki


Tue 14, 18:33
Text Editor: Code for Steve
Code snippet that Steve asked for:
**** code starts here ****
**** code ends here ****


Tue 14, 18:33
Text Editor: Code for Steve
You have asked to convert this document into Plain Text.
If confirmed, the conversion will discard all formatting in this document.
Convert to Plain Text
Code snippet that Steve asked for:
**** code starts here ****
**** code ends here ****


Tue 14, 18:43
Internet Browser: Cute Overload!
cute
OVERLOAD
Mom, Ellie and I are going to do a routine
A slice of heaven on your desk!
Check it out

4.2 Test goals

Because of the limited feature set in the prototype, the user test was focused on the most important aspects of the user interface (with the exception of scrolling, which would have required capturing right-click mouse events — an action not supported in the Flash runtime environment). The tested features included:

- Launching applications
- Understanding of the applet strip
- Navigation (within the application as well as the overview layer)
- Issuing commands (via the menu structure and/or search functionality)
- Adaptive highlighting and prioritizing of predefined elements
- Simple window management (reorganizing application windows)

Not included in the test were more advanced user interface features designed for aiding in prolonged use of the system, or geared towards power users:

- Advanced window management (creating groups, merging windows)
- Console functionality (application-specific command line interface)
- The adaptive nature of command strips and the launcher screen
- Metadata manipulation
- Scrolling (due to runtime limitations)

While not complete, the provided feature set proved extensive enough for a meaningful preliminary user test. Adding more functionality to the prototype would have allowed for a longer, more pronounced user test, but also increased the development time significantly. Since Stripes is first and foremost a design concept, it seemed that testing some of the key functionality as early as possible would yield significant feedback despite the lack of some of the more advanced features.

4.3 Testing procedure

The user test was designed as a sequential task list, where testers would perform given tasks step-by-step. The test included a total of 12 tasks, ranging from simple logging into the system to copying snippets of text into a newly created document and converting it to plain text format. The overall difficulty of the tasks could be classified as "easy", especially since the prototype only contained a handful of applications, files, web locations, and commands. The complete task list can be found in the Appendix B. Before running any actual tests, a pilot run of the test was carried out with a person unfamiliar with the project in order to pinpoint any problems with the tasks or their wording.

Because of the mobility of the test platform, the tests could be conducted at multiple locations where necessary. Prior to the actual test, the testers were given some background information on the project in general, but nothing about the actual interface was revealed. As users were conducting the test, notes were taken on their actions. Since the prototype had several bugs in it, testers were asked to repeat certain actions or tasks if any technical problems occurred during the test. After completing the tasks, all testers were asked to provide additional, freeform feedback, and were inquired to elaborate on any issues that arose during the test. During the test, users were encouraged to speak out loud, explaining what they were seeing and doing, and why. An audio track, a video capture of the screen, as well as a video of the face of the tester were recorded, where possible. Figure 45 presents a screenshot of a user test video. Despite recording, most users felt — according to their own words — that the testing situation was calm and relaxed. Even more anxious testers seemed to quickly overcome any initial signs of stress and anticipation, such as heavy breathing or a tense posture once they started to complete the tasks.

A total of 13 test runs were carried out in a timeframe of approximately two weeks. The testers were in the age range of 20–36 years old, comprising of 4 females and 9 males with varied computer experience. Unsurprisingly for the age group, no computer novices were included, as everyone felt they had at least mediocre computer skills; several participants, on the other hand, described themselves as power users with up to 20 years of experience with computer user interfaces. None of the testers had any prior experience with the *Stripes* prototype, however, and did not know any details about the user interface beforehand. It is unfortunate that the user group did not include any less-experienced users, calling for additional test rounds in the future.

The user test in itself was another learning experience. Designing, organizing, and executing the user test sessions was not particularly challenging, but in hindsight several aspects of the testing situation could have been improved. The tasks of the test, for example, seemed almost banal to some testers in their simplicity; the prototype should have contained much more dummy content to increase the difficulty of locating files and applications. Also,

known bugs in the prototype unnecessarily hindered the testing experience for some users. The very nature of the test did not yield any quantitative data, and since there was only one test round, the true learning curve of the prototype can only be hinted at.

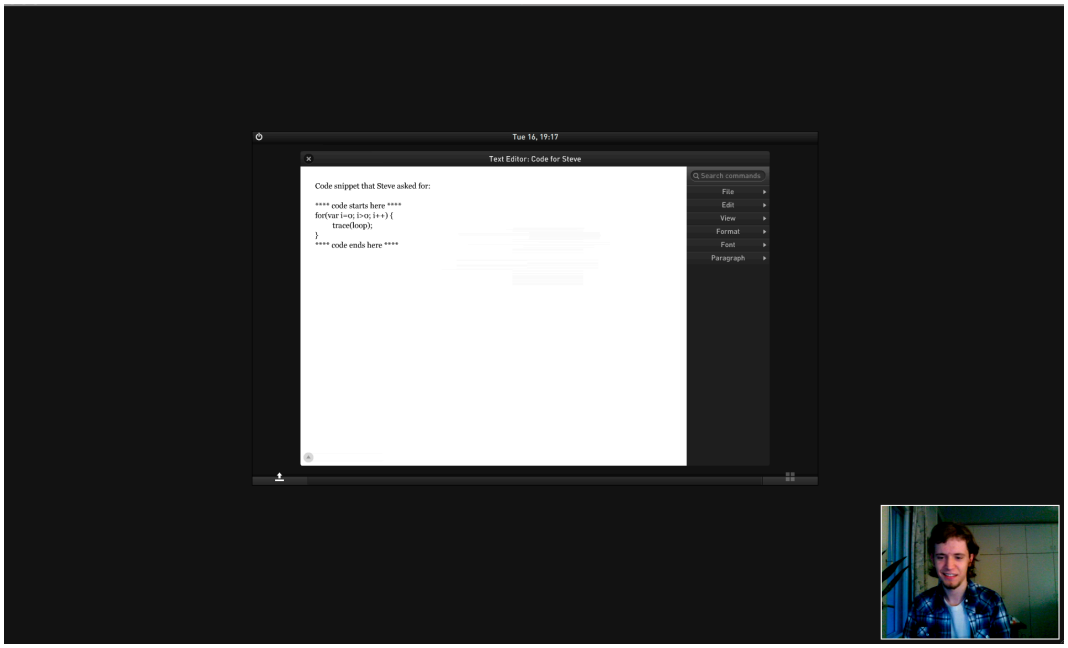


Figure 45 **A user participating in the test. Video of both the screen and the user as well as audio are being recorded simultaneously.**

4.4 Test results

Because of the nature of the test being sequential and not timed, the results of the test are mostly qualitative. During the feedback session after the actual test had been completed, some participants provided their insights on what features they felt were missing in the prototype, and suggested other improvements to the system. Often, the requested features were simply missing in the prototype or had been omitted due to technical limitations of the platform. Some of the proposed changes did not resonate well with the vision of the project, described in more detail in subsection 1.2.3, and where therefore set aside; fitting suggestions are listed in subsection 4.4.3.

The test results have been compiled into two subsections, 4.4.1 and 4.4.2, explaining in detail the positive and negative test feedback, respectively. Suggestions on how to improve the user interface concept for its next iteration are also provided.

4.4.1 Favorable outcomes

Generally speaking, the prototype received very positive feedback. Every test user was able to complete all of the tasks involved (a total of 156 successful completions), excluding a few situations where bugs temporarily hindered the accomplishment of a given task. Most test users seemed to enjoy the experience, and no clear signs of frustration were evident during or after the test.

Below is a compiled list of the favorable outcomes of the user test. They are sorted in the order of their frequency of occurrence.

- Despite initial confusion over some of the elements in the user interface, as explained in more detail in subsection 4.4.2, the system seemed to promote a low learning curve: once users had successfully tried out a button or another interface element, they immediately learned its functionality and memorized it (at least for the duration of the test). This applied to all participants.

- The use of contextual commands, appearing in the command strip when an element was selected or highlighted, was very successfully used by all testers. The immediate and decisive response to locate and select a contextual command was noticeable during the test, underlining the high usability and efficiency of the design. Several users mentioned the use of color in the contextual commands as a further visual aid in locating them. Two testers tried out right-clicking the mouse for a contextual menu at first, a habitual response according to their own words, but quickly proceeded with using the command strip after realizing the contextual menu was not working as expected (for technical reasons, the summoned contextual menu was that of the prototype application framework, not the user interface prototype being tested). Figure 46 shows a test user using the contextual menu system.

- The drag-and-drop functionality in the overview mode seemed intuitive to all but one participant. Some testers felt a bit unsure whether or not such an “advanced feature” could actually work, but tried it out regardless — much to their delight. This is a surprisingly positive result, as the feature could easily be further enhanced with additional feedback and other visual cues. All users were also able to zoom back into a given window after rearranging them. Figure 47 presents a test user successfully dragging a window to rearrange the window row.

- While looking for a way to see all open windows simultaneously, the overview button in the common UI frame was easily identified by eight of the thirteen (62%) testers because of its comprehensible iconic appearance that seemed to match its expected

Figure 46 A test user using the contextual menu system to copy a snippet of code.

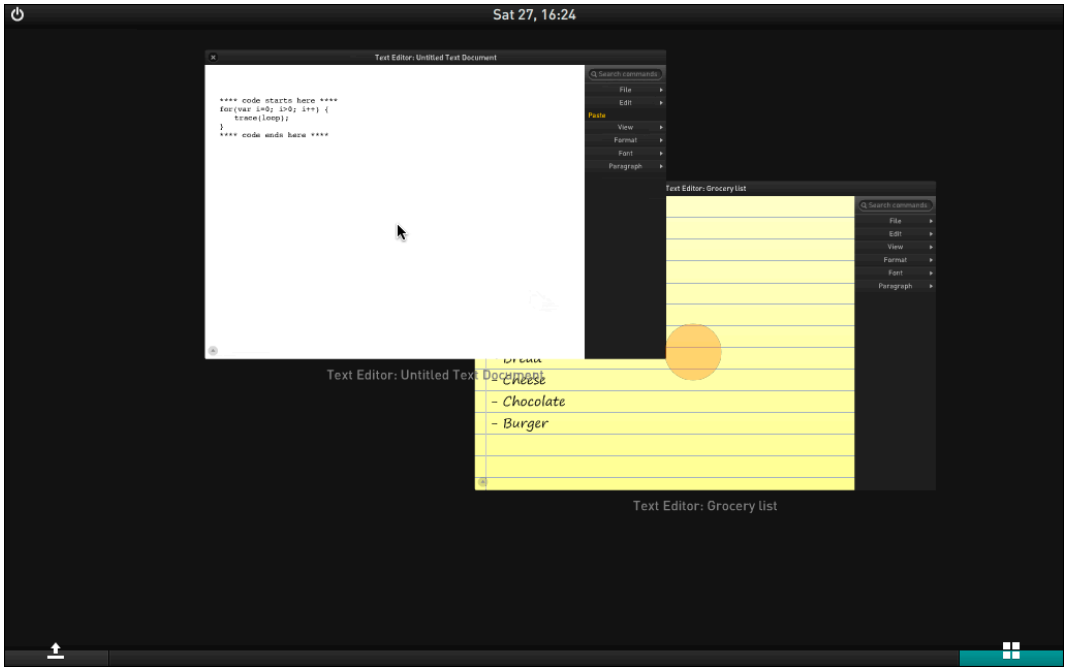


Figure 47 A test user rearranging the window strip.

behavior. This was further reinforced by the descriptive tooltip that was displayed when the cursor moved over the button.

- While not a specific question during the test, six out of thirteen participants (46%) explicitly mentioned that they liked the overall look-and-feel of the user interface; nobody raised direct criticism against the graphical design of the prototype, but one user mentioned his concern about visual flicker (animation) when using the system for an extended period of time. Another user applauded the restricted use of color, noting that he immediately focused on the active launcher button after booting up the system because it "stood out from the rest of the environment". Yet another user concluded that in the prototype, locating interface elements was easy because they were visible and arranged in an uncluttered manner.

- Two test participants (15%) explicitly mentioned that, after using the prototype for only 10–15 minutes, they felt a sense of mastery while using the system.

- Two power users (15%) used the action bar in order to enter a web location, despite the fact that the Internet browser application was visible and readily available in the launcher. Another user mentioned that he considered using the bar, but had reservations of doing so because of his prior experiences with CLI-style launchers that did not support entering URLs.

- The use of text over icons did not raise much feedback, but one user (8%) mentioned his dissatisfaction over the current trend towards heavy icon usage in user interfaces, stating that it has become difficult to recognize and locate commands because of it.

Another generally observed favorable outcome was the formation of the users' correct conceptual model of the user interface. In the case of the user test, nearly all of the participants had, by the end of the test, formed a conceptual model that seemed fairly consistent with the designer's model. This became apparent once testers had successfully tried out the most important interface elements; a clear understanding of the general concept of launching processes, the row-like window arrangement, as well as navigating and managing the workspace was evident, and the fluent use of the interface (with the exception of certain poorly implemented features, described in more detail in subsection 4.4.2) underlined the fact that participants had grasped and were mastering the system. Because the overall design of Stripes has been modeled primarily after existing user interfaces, and doesn't represent a radical departure of established norms, most participants were able to apply their existing conceptual models to the new interface and trust it to behave similarly, fortifying their sense of control and comprehension. When new models emerged, such as with the contextual use of commands, the lack of a Desktop, or arranging windows, users

seemed to be surprisingly able to change their existing model and quickly adopt to the new environment.

Users' habits and their conceptual models based on prior experiences with computer user interfaces were visible during several tasks. Many users, for example, rearranged windows by dragging their title bar (although dragging could be established by dragging any part of the application window), activated contextual menus, or double-clicked on files and applications to open them (although a single-click would have sufficed). As described in subsection 4.4.2, some users missed familiar features and interface elements from other operating systems, but their absence did not hinder them in accomplishing the tasks of the test — further underlining the fact that they were able to quickly adapt to the mental model of *Stripes*.

4.4.2 Shortcomings

Despite the overall success of the user tests, there were a handful of shortcomings in the implementation and design of the prototype that caused confusion and user errors. Also, bugs in the interface caused unnecessary uncertainty in some of the users, albeit only temporarily.

Below is a compiled list of the shortcomings of the proposed design and its current implementation. They are sorted in the order of their frequency of occurrence.

- Nine out of thirteen users (69%) were initially confused by the two large, inactive buttons of the common UI frame. Especially the launcher button that was simultaneously highlighted yet inactive attracted testers to unnecessarily click it. It was clear that testers did not associate the launcher button with the launcher screen, although once they had successfully used the button later on during the test, its meaning and function quickly became understandable. The initial confusion never caused any actual errors, but any mis-clicks should be avoided nevertheless. Figure 48 shows a test user clicking on the inactive overview button.

- Once multiple windows had been opened simultaneously, the previous and next windows in the window row were partially visible whenever a window was in focus. Because of technical restrictions, these windows had no visual cue on their inactive or unfocused state (unlike originally planned). Six testers (46%) found this to be distractive or confusing, and only four (31%) understood that they could be clicked to move within the window row. Especially the window to the left caused confusion, since none of its contents but rather a part of its command strip could be seen. This particular

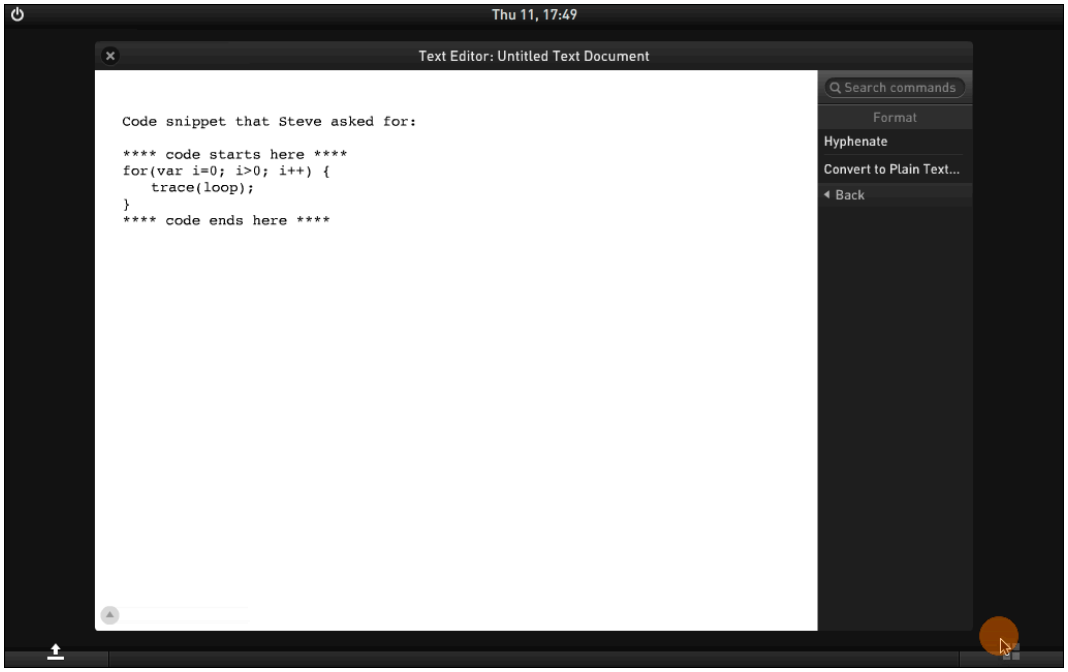


Figure 48 A test user trying to click on the inactive overview button.

problem did not cause any errors per se, but was the most-critiqued visual aspect of the prototype.

- Six participants (46%) noticed the welcome screen button hovering on the lower left corner of each application window, and all of them found the technical implementation of the button to be confusing; when hovering the mouse cursor over the button, the header of the welcome screen (titled "Springboard" in the prototype) peaked into view. Once visible, users tried to move along the bar towards the header titles in an effort to click them, but this action pulled the bar back, instead, effectively hiding the bar. This shortcoming caused the most frustration in users, and therefore needs to be addressed as a top priority in a future iteration of the prototype.

- Four users (30.8%) did not notice the action bar located in the launcher screen at all during the test. When questioned about it after the test, users noted that the launcher simply had too many other interface elements that competed for their attention. Since promoting textual input is one of the goals of the proposed design due to its efficiency, this problem is noteworthy.

- Albeit all test participants were able to browse the command strip, three users (23%) felt that browsing was slow in relation to classic menu bars. Two users (15%) tried to use the group headers inside the submenus to return to the "front page" of the command strip, and another two were confused by the labeling of the "back"-button

in the submenus, using them in an effort to return to a previous application window. None of the test users used the command search function; three testers (23%) noted its subdued visual look to be a reason why it felt inactive, others mentioned their unfamiliarity of using the keyboard in general. Although all users eventually mastered the command strip without any mis-clicks, the design was clearly not optimal, and calls for improvement.

- Two testers (15%) wanted to click on the launcher column headers in order to maximize them, despite noting that they didn't look like they could be activated in any way. These users later asked about the resizing possibility of the columns (a feature not supported in the prototype). Although not a crucial issue in this particular user test, future prototypes with more content will call for column resizing functionalities.

- Another two users (15%) were perplexed about the need for manually activating text fields after new text documents were created or a text document was focused on. This behavior was not designed per se, but rather a compromise forced by the technical implementation of the contextual menus. It could thus be classified as a bug in the prototype; regardless, the problem seemed to cause much frustration for those two testers who encountered it, and should be considered top priority for future iterations of the prototype.

- Yet another two testers (15%) were hoping to be able to close windows in the overview mode, a feature that ironically was cut from the prototype very late in the development process. Three testers (23%) also missed the Windows Taskbar for quickly switching between open windows. Feature-requests such as these are not necessarily crucial, since both actions (closing windows and navigation) can be accomplished by other means; as with all features, their benefits will have to be carefully evaluated against the vision of the project, as feature bloat is never a desired side effect.

Although not explicitly mentioned during the user test, one of the most questionable aspects of the design is the command strip and the large horizontal space it requires. While not a problem on larger resolutions, on smaller screens the design of the strip might end up doing more harm than good. One possible solution includes hiding the strip on small screens, but this resolution undermines the strip's fundamental benefit of being constantly visible.

4.4.3 Suggested improvements based on the test

The major shortcomings of the prototype that became evident during the preliminary user test help in forming a consensus on what areas of the prototype require the most fixing for its next iteration. These include:

- Visually tweaking the launcher and overview buttons in such a manner that they look more subdued when inactive, reducing their affordance and attraction to clicks. The launcher icon could also use some improvement.

- Visually tweaking the launcher screen so that it feels more connected with the launcher button, when active. This is an effort to immediately convey the notion of the launcher screen being associated with its respective button.

- Adding visual cues to the unfocused windows, and possibly some indication that they can be clicked. Because of the rather uninformative nature of the previous window in the window row (which only displays a part of its command strip), the design of the feature will need to be reviewed and possibly revised.

- A complete redesign of the welcome screen button and its functionality.

- Visually tweaking the action bar to be more prominently visible in the launcher screen.

- Visually lifting the command search bar in an effort to promote its use, and renaming the “back”-button in the command strip submenus in order to avoid confusion with the Internet browser’s “back”-button.

- Fixing all found bugs in the prototype.

Despite the list of improvements, the fundamental design approaches that were tested seem to be in relatively good order. It should also be noted that none of the shortcomings completely hindered the use of the prototype or caused tasks to become unachievable, as most of them seem to be rectifiable with tweaking the visuals of the user interface.

There were also a handful of ideas presented by the test users in order to improve the design of the proposed interface design. The most fitting suggestions are listed below:

- The highlight color of the launcher and overview buttons in the common UI frame could reflect the overall highlight color used with files and contextual commands.

- The welcome screen button hover functionality could show more content of the welcome screen than just the titles of columns, allowing for quickly launching recent files or creating new ones.[53]

- Allowing the user to tag window rows with metadata, and then presenting them as a stack in the launcher window for quickly opening a set of application windows with one click.

- Allowing for customizing the bottom middle part of the common UI frame between the launcher and overview buttons to include shortcuts to applications, files, websites, contacts, etc. Since this feature would add complexity to the user interface, it should be considered a non-default option only.

- Treating the launcher screen as a window instead of a panel, thus being directly accessible from the overview mode, as well.

All of these aforementioned suggestions could be added to the original design without the need for fundamentally revising the concept, and could therefore be tested with alternative prototypes.

4.5 Reflection on focus areas

In addition to highlighting the strengths and weaknesses of the design concepts as they were presented in the prototype, the user test results can also be reflected on the original goals of the project, namely the eight focus areas outlined in subsection 1.2.2. Unfortunately, because of the limited nature of the test, the achievement of *all* of the aforementioned design goals cannot be adequately analyzed yet; an improved, more full-featured prototype would be required for a thorough study.

One of the most prominent goals of the concept is to reduce complexity and management of operating system user interfaces. Simplicity has been achieved with the omission of many user interface elements for managing application windows, the clear division between launching processes and navigation, the inclusion of a reliable, organized way to see the complete overview of the workspace, and the intuitive window management features. The user test results corresponded with this design goal, as users had little trouble finding interface elements and working their way around the workspace; several users noted that they were able to find functionalities simply because there was so little interface clutter.

53 The use of hover actions in general is somewhat controversial, since they cannot currently be triggered by touch interfaces.

The reduction of management, on the other hand, is largely achieved with the use of adaptive interface solutions, such as the concept of popularity — a feature not included in the prototype.

Another high priority of the proposed user interface concept is to increase usability and user satisfaction. While difficult to measure in a qualitative test, usability seemed to score quite well overall. If measured against Nielsen's five usability criteria, as presented in subsection 1.2.2, the test results suggest that the concept had high learnability and memorability, as users immediately understood and later remembered what all the various interface elements were used for; an exception were the launcher and overview buttons that caused *initial* confusion, but even their functionality was quickly grasped after they became active. Nevertheless, more thorough user tests will have to be conducted before definitive results on learnability and memorability can be presented. Error rates and general efficiency were not quantitatively tested. User satisfaction was surprisingly high, as several users spontaneously said that they had had "a good time" and even "fun" during the test and did not regret taking it.

The visual attractiveness of the system, a highly important design goal, received good feedback overall. While not every user commented on the look-and-feel of the prototype, everyone who did spoke highly of it; it is still unclear whether or not the high contrast of the user interface will become tiring to the eyes over extended periods of time, however. Also, there is a clear need for some improvement in the visuals in terms of increasing usability and affordance, such as in the unfocused application windows.

Since the user test group was not diverse enough to cover the whole target group, as presented in section 1.3, and since all of the power-user features such as keyboard shortcuts, the console, and a fully functional action bar were missing, it is difficult to say how well the proposed interface design resonates with novice and power users. It has to be noted that none of the testers who considered themselves as power (or advanced) users felt that they were being unproductive or inefficient; nobody seemed to be insulted by the simplicity of the prototype.

The overall efficiency of the user interface was not directly tested or measured, nor was the scalability to various screen resolutions or input devices. Naturally, the achievability of the system in terms of technology remains only a theory, although the fact that a prototype for testing purposes could be built with unexceptional programming skills and in a reasonably short time period is a good sign.

5 Conclusions

The *Stripes* user interface project started out as an effort to research the current status quo of operating system user interfaces, their respective strengths and weaknesses, history, and relevance in today's world of computing. In addition, the task of creating a revised interface design was undertaken, largely based both on the theoretical work done by some of the most respected usability experts in the field, as well as the recent practical work carried out by visionary individuals, teams, and corporations. Finally, a limited prototype of the proposed interface design was created and tested, as well as evaluated against the original design goals.

Based on the preliminary user test results, the fundamental design approach of *Stripes* was welcomed and applauded; certainly, several issues arose during the test, requiring more iterative work to be done on the prototype, but the outcome was favorable enough to promote continuing effort on the improvement of the design. However, as some features of the design have not been tested yet, and many require continuous use of the system for an extended period of time, the prototype needs to advance significantly and be tested thoroughly before any definite validation can take place.

Even if this particular interface design fails to materialize or is deemed unsuccessful, the recent developments in the field of personal computing all point to a longing for simplifying and modernizing the fundamentals of the operating system user interface designed decades ago. Computerized appliances, capable of achieving many of the same functionalities as traditional desktop and notebook computers, have provided a reference to just how unnecessarily complicated and ineffective the current OS user interface implementations have grown to become; the Internet, now able to host simplistic yet feature-rich applications, has revived the network-centric design approach of thin clients and is unstoppably changing the way humans interact, collaborate, and work together — both on small-scale mobile devices as well as on conveniently large home or office systems.

Even if the web browser is gradually becoming the most important application for everyday computing, there still remains the need to fluently organize and manage our digital workspace, even if in the future, applications, files, and other content is mainly stored and accessed via the Internet. This necessity is at the heart of concepts such as *Stripes*, an effort to streamline and improve the overall user experience for the requirements of today's and tomorrow's computing needs, without radically departing from the respected, proven heritage of the past.

On a personal level, the project provided a unique opportunity for me to focus on and research the history and development of operating system user interfaces, and inquire into many mind-expanding theoretical works written by well-respected experts in the field. It

also allowed for unconstrained experimentation in user interfaces, especially in a context that would not have been possible to achieve under typical conditions in the workplace. The included design, programming, and graphical work, as well as testing the concept further accounted for a tremendous learning experience overall. With the help of all the feedback and positive reinforcement, the continuation of the project is a formidable possibility — if not directly, at least in spirit with any possible future concepts. For someone passionate about working on interaction, user interface, or software design, there could hardly have been a greater opportunity to dream up, prototype, and test a project of this size and ambition.

References

Publications

Foley, James D.; van Dam, Andries; Feiner, Steven K.; Hughes, John F.; Phillips, Richard L. *Introduction to Computer Graphics*. Addison-Wesley Publishing Company. 1994 (1990).

Johnson, Steven. *Interface Culture*. Basic Books. 1997.

Nielsen, Jacob. *Usability Engineering*. Academic Press, Inc. 1993.

Norman, Donald A. *The Design of Everyday Things*. Doubleday. 1990 (1988).

Norman, Donald A. *Emotion & Design: Attractive things work better*. ACM Interactions. July + August, 2002.

Norman, Donald A. *The Invisible Computer: Why Good Products Can Fall, the Personal Computer Is So Complex, and Information Appliances Are the Solution*. The MIT Press. 1998.

Preece, Jenny; Rogers, Yvonne; Sharp, Helen; Benyon, David; Holland, Simon; Carey, Tom. *Human-Computer Interaction*. Addison-Wesley Publishing Company. 1994.

Raskin, Jef. *The Humane Interface*. ACM Press. 2000.

Shneiderman, Ben. *Designing the User Interface. Strategies for Effective Human-Computer Interaction. 2nd Edition*. Addison-Wesley Publishing Company. 1992, reprinted with corrections October, 1993.

van der Veer, Gerrit C.; Puerta Melguizo, Maria del Carmer. *Mental Models*. Published in: Jacko, Julie A.; Sears, Andrew (eds.). *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Lawrence Erlbaum Associates. 2002. Referenced through: Brinck, Anna. *Käyttäjän käsitemallien merkitys matkapuhelimen käyttöönotossa*. Pro gradu -tutkielma. Tampereen yliopisto, tietojen-käsittelytieteiden laitos, 2005.

Wardrip-Fruin, Noah; Montfort, Nick (eds.). *New Media Reader*. The MIT Press, 2003.

Ware, Colin. *Information Visualization: Perception for Design, 2nd Edition*. Elsevier Inc. 2004.

Online references

Agarawala, Anand; Balakrishnan, Ravin. *BumpTop*. http://bumptop.com/ [accessed: December 4, 2007]

Apple Support. *Mac 101: Automator*. December 18, 2007. http://docs.info.apple.com/article.html?artnum=304759 [accessed: January 23, 2008]

Apple Support. *Mac 101: Exposé*. December 18, 2007. http://docs.info.apple.com/article.html?artnum=304786 [accessed: December 22, 2007]

Bangaru, Vijay. *Welcome*. WinFS Team Blog. August 29, 2005. http://blogs.msdn.com/winfs/archive/2005/08/29/457615.aspx [accessed: January 7, 2008]

Barkhuus, Louise; Dey, Anind. *Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined*. Proceedings of UbiComp 2003, pages 150–156, Springer. 2003. http://www.itu.dk/people/barkhuus/barkhuus_ubicomp.pdf [accessed: January 4, 2008]

Bush, Vannevar. *As We May Think*. The Atlantic.com. Originally published in the Atlantic Monthly, June 1945. http://www.theatlantic.com/doc/194507/bush/ [accessed: January 23, 2008]

Carter, Phillip R. *A Brief History of GUI*. June 21, 2005. http://cognitivevent.com/gui_history.html [accessed: January 24, 2008]

DiCarlo, Jono; Raskin, Aza; Varma, Atul; Wilson, Andrew. Humanized.com. *About Us*. http://www.humanized.com/about/ [accessed: December 28, 2007]

Engelbart, Douglas. *Augmenting Human Intellect*. Stanford Research Institute. October 1962. http://www.invisiblerevolution.net/engelbart/full_62_paper_augm_hum_int.html [accessed: January 23, 2008]

Folklore.org. *Creative Think*. Based on notes by Andy Hertzfeld from Alan Kay's speech at the Creative Think seminar, July 20, 1982. http://www.folklore.org/StoryView.py?project=Macintosh&story=Creative_Think.txt&sortOrder=Sort%20by%20Date&detail=medium&search=kay [accessed: November 20, 2007]

Goodger, Ben. *Why the Mozilla Browser Sucks*. Inside Firefox. January 25, 2003. http://weblogs.mozillazine.org/ben/archives/005239.html [accessed: November 20, 2007]

Goodger, Ben. *Where Did Firefox Come From?* Inside Firefox. February 6, 2006. http://weblogs.mozillazine.org/ben/archives/009698.html [accessed: November 20, 2007]

Hale, Kevin. *Visualizing Fitt's Law*. Particletree. October 3, 2007. http://particletree.com/features/visualizing-fittss-law/ [accessed: January 8, 2008]

Harris, Jensen. *It's All About Context*. An Office User Interface Blog. September 16, 2005. http://blogs.msdn.com/jensenh/archive/2005/09/16/468365.aspx [accessed: January 1, 2008]

Harris, Jensen. *The Long Road to Contextual Tabs*. An Office Interface Blog. March 7, 2006. http://blogs.msdn.com/jensenh/archive/2006/03/07/545300.aspx [accessed: January 1, 2008]

Harris, Jensen. *Combating the Perception of Bloat (Why the UI, Part 3)*. An Office User Interface Blog. March 31, 2006. http://blogs.msdn.com/jensenh/archive/2006/03/31/565877.aspx [accessed: January 1, 2008]

Harris, Jensen. *Let's Talk About Customization*. An Office User Interface Blog. June 27, 2006. http://blogs.msdn.com/jensenh/archive/2006/06/27/648269.aspx [accessed: January 4, 2008]

Henderson, Jr.; Card, Stuard. *Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface*. Published in: ACM Transactions on Graphics (TOG), Volume 5, Issue 3 (July 1986), ACM Press, p. 211–243. Provided online by ACM Portal: http://portal.acm.org/citation.cfm?id=24056 [accessed: October 5, 2008]

Holwerda, Thorn. *Common Usability Terms, pt. III: Desk Accessories*. OSNews.com. November 11, 2007. http://www.osnews.com/story.php/18875/Common_Usability_Terms_pt._III:_Desk_Accessories [accessed: December 30, 2007]

Holwerda, Thorn. *Common Usability Terms, pt. VI: the Dock*. OSNews.com. November 18, 2007. http://www.osnews.com/story.php/18941/Common-Usability-Terms-pt.-VI-the-Dock/page1/ [accessed: December 16, 2007]

Holwerda, Thorn. *Common Usability Terms, pt. VIII: Tabs*. OSNews.com. August 16, 2008. http://www.osnews.com/story/20191/pt_VIII_Tabs [accessed: September 24, 2008]

Kapelonis, Kostis. *Project Elevate*. http://elevate.sourceforge.net/index.html [accessed: December 4, 2007]

Knowing.NET. *How Many Lines of Code in Windows?* Larry O'Brien, quoting Vicent Maraia's book *The Build Master: Microsoft's Software Configuration Management Best Practices*. December 6, 2005. http://www.knowing.net/PermaLink,guid,c4bdc793-bbcf-4fff-8167-3eb1f4f4ef99.aspx [accessed: November 20, 2007]

Lineback, Nathan. *Graphical User Interface Gallery*. 2007. http://toastytech.com/guis/index.html [accessed: January 24, 2008]

Microsoft Developer Network. *Desktop Window Manager*. http://msdn2.microsoft.com/en-us/library/aa969540.aspx [accessed: December 30, 2007]

Microsoft Office Labs. *Search Commands*. November 6, 2007. http://www.officelabs.com/projects/searchcommands/Pages/default.aspx [accessed: June 12, 2008]

Microsoft Research. *The TaskGallery*. July 28, 2005. http://research.microsoft.com/adapt/TaskGallery/ [accessed: January 11, 2008]

Opera Software. *Mouse Gestures in Opera*. http://www.opera.com/products/desktop/mouse/ [accessed: January 14, 2008]

Quinn, Ryan. *SymphonyOS Project*. http://www.symphonyos.com/ [accessed: December 4, 2007]

Raskin Center for Humane Interfaces. http://rchi.raskincenter.org/ [accessed: December 4, 2007]

Raskin, Aza. *One Thing at the Time and the Multitasking Myth*. Humanized Weblog. June 8, 2006. http://www.humanized.com/weblog/2006/06/08/one_thing_at_a_time_and_the_mutlitasking_myth/ [accessed: January 12, 2008]

Raymond, Eric Steven; Landley, Rob W. *The Art of Unix Usability*. 2004. http://www.catb.org/~esr/writings/taouu/html/index.html [accessed: January 29, 2008]

Reimer, Jeremy. *A History of the GUI*. Ars Technica. May 5, 2005. http://arstechnica.com/articles/paedia/gui.ars/ [accessed: January 23, 2008]

Ross, Blake. *The Firefox Religion*. January 22, 2005. http://blakeross.com/2005/01/22/firefox-religion/ [accessed: November 20, 2007]

Sellen, Abigail; Kurtenbach, Gordon; Buxton, Bill. *The Prevention of Mode Errors Through Sensory Feedback*. Human-Computer Interaction, Vol. 7(2), p. 141-164. 1992. http://www.billbuxton.com/ModeErrors.html [accessed: January 11, 2008]

Selvidge, Paula. *Menu Design: To Adapt, or Not to Adapt?* Software Usability Research Laboratory, Wichita State University. Usability News. January 2002, Vol. 4 Issue 1. http://www.surl.org/usabilitynews/41/adapt_menus.asp [accessed: June 6, 2008]

Shayo, Conrad; Olfman, Lorne. *The role of conceptual models in formal software training*. Published in: Proceedings of the 1998 ACM SIGCPR conference on Computer personnel research, 1998, ACM Press, p. 242–253. Provided online by ACM Portal: http://portal.acm.org/citation.cfm?id=279179.279216&coll=Portal&dl=ACM&CFID=5500438&CFTOKEN=25402612 [accessed: October 7, 2008]

Spisak, Jason. *Mezzo User Interface — Principles of a Simplified Desktop User Interface and Usability Experience*. 2004-2005. http://ftp.sun.ac.za/ftp/iso-images/mezzo/MezzoGreypaper.pdf [accessed: December 4, 2007]

Stephenson, Neal; Birkel, Garrett. *The Command Line in 2004*. Based on the essay: *In the Beginning… was the Command Line* published in 1999 by Neal Stephenson, annotated with the author's permission by Garrett Birkel. July 5–December 29, 2004. http://garote.bdmonkeys.net/commandline/index.html [accessed: January 29, 2008]

Sun Microsystems. *Project Looking Glass*. http://www.sun.com/software/looking_glass/ [accessed: December 4, 2007]

The Linux Information Project. *The X Window System – A Brief Introduction*. March 29, 2006. http://www.linfo.org/x.html [accessed: December 22, 2007]

Tognazzini, Bruce. *Top ~~Ten~~ Nine Reasons the Apple Dock Still Sucks*. AskTog.com. January 2004. http://www.asktog.com/columns/044top10docksucks.html [accessed: December 16, 2007]

van Dam, Andries. *Post-WIMP User Interfaces*. Communications on the ACM, Volume 40, Nr 2. February 1997. http://citeseer.ist.psu.edu/cache/papers/cs/11222/http:zSzzSzcs.ru.ac.zazSzhomeszSzg97rc001zSzpaperszSzp63-van_dam.pdf/dam97p-ostwimp.pdf [accessed: January 2, 2008]

W3Schools. *Browser Statistics by the Month — September 2007*. http://www.w3schools.com/browsers/browsers_stats.asp [accessed: November 20, 2007]

Weeks, Alex. *Linux System Administrator Guide*. Version 0.9. 2004. http://www.tldp.org/LDP/sag/html/sag.html#GNU-OR-NOT [accessed: January 24, 2008]

Figures

Figure 1 Unknown. Via: The Institute for the Future of the Book. http://www.futureofthebook.org/blog/archives/memex-1.jpg [accessed: October 5, 2008]

Figure 2 Standford University. Screenshot from a video demonstration. Via: MouseSite. http://sloan.stanford.edu/MouseSite/1968Demo.html [accessed: October 5, 2008]

Figure 3 Xerox PARC. Via: Ars Technica. http://media.arstechnica.com/images/gui/7-AltoST.jpg [accessed: October 5, 2008]

Figure 4 University Video Communications. Composite of four screenshots of a video demonstration. Via: Internet Archive and Wikipedia. http://upload.wikimedia.org/wikipedia/en/7/7b/Sketchpad-Apple.jpg [accessed: October 5, 2008]

Figure 5 Apple Incorporated. Via: GUIdebook, Graphical User Interface Gallery. http://www.guidebookgallery.org/pics/gui/desktop/full/macos11.png [accessed: October 5, 2008]

Figure 6 Amiga Corporation. Via: GUIdebook, Graphical User Interface Gallery. http://www.guidebookgallery.org/pics/gui/desktop/full/amigaos10.png [accessed: October 5, 2008]

Figure 7 Acorn Computers. Via: Ars Technica. http://media.arstechnica.com/images/gui/20-riscos.gif [accessed: October 5, 2008]

Figure 8 Be Incorporated. Via: GUIdebook, Graphical User Interface Gallery. http://www.guidebookgallery.org/pics/gui/desktop/full/beos5.png [accessed: October 5, 2008]

Figure 9 Apple Incorporated. Via: GUIdebook, Graphical User Interface Gallery. http://www.guidebookgallery.org/pics/gui/desktop/full/macosxpb.png [accessed: October 5, 2008]

Figure 10 Auramo, Helene. Screenshot of Mac OS X, courtesy of Apple Incorporated. September 26, 2008.

Figure 11 Hale, Kevin. October 3, 2007. Via: Paricletree. http://particletree.com/examples/fitt/f8.gif, http://particletree.com/examples/fitt/f4.gif [accessed: October 5, 2008]

Figure 12 Raskin Center for Humane Interfaces. October 7, 2005. http://rchi.raskincenter.org/images/4/4b/Screenshot_startpage.gif [accessed: October 5, 2008]

Figure 13 Agarawala, Anand; Balakrishnan, Ravin. http://www.bumptop.com/temp_images/mainimage_new.jpg [accessed: October 5, 2008]

Figure 14 Microsoft Corporation. Via: Toasty Tech. http://toastytech.com/guis/bobhome1p.png [accessed: October 5, 2008]

Figure 15 Kapelonis, Kostis. http://elevate.sourceforge.net/img/central_hub.full.jpg [accessed: October 5, 2008]

Figure 16 Sun Microsystems. http://www.sun.com/images/k3/k3_lglass_gallery3.gif [accessed: October 5, 2008]

Figure 17 Spisak, Jason; Quinn, Ryan. April 2, 2008. http://symphonyos.com/screenshots/SymphonyOne-ss2.png [accessed: October 5, 2008]

Figure 18 (a) Acorn Computers. Via OSNews.com. http://www.osnews.com/img/18941/risc4.png [accessed: October 5, 2008] (b) The Open Group. Via OSNews.com. http://www.osnews.com/img/18941/cde.gif [accessed: October 5, 2008] (c) Fourdan, Olivier, et al. Via OSNews.com. http://www.osnews.com/img/18941/xfce.jpg [accessed: October 5, 2008] (d) Apple Incorporated. Via: GUIdebook, Graphical User Interface Gallery. http://www.guidebookgallery.org/pics/gui/desktop/firstrun/macosx103.png [accessed: October 5, 2008]

Figure 19 Microsoft Corporation. Via: Sunflowerhead.com. http://www.sunflowerhead.com/msimages/AdaptiveMenus-10-8-2005.png [accessed: October 5, 2008]

Figure 20 Apple Computer. http://www.apple.com/findouthow/mac/#tutorial=expose [accessed: October 5, 2008]

Figure 21 Opera Software. http://www.opera.com/products/desktop/mouse/ [accessed: October 5, 2008]

Figure 22 Gimpl, Martin. Screenshot of the Mozilla Firefox browser, courtesy of Mozilla community. October 5, 2008.

Figure 23 (a) Novell Incorporated. Via: Wikipedia. http://upload.wikimedia.org/wikipedia/en/a/a2/OpenSUSELinuxVD.png [accessed: October 5, 2008] (b) Canonical Ltd. / Ubuntu Foundation. Via: Heise Online. http://www.heise-online.co.uk/images/110025/2/1 [accessed: October 5, 2008] (c) Apple Incorporated. Image edited: background color added. http://images.apple.com/macosx/features/images/spaces_hero20071016.png [accessed: October 5, 2008]

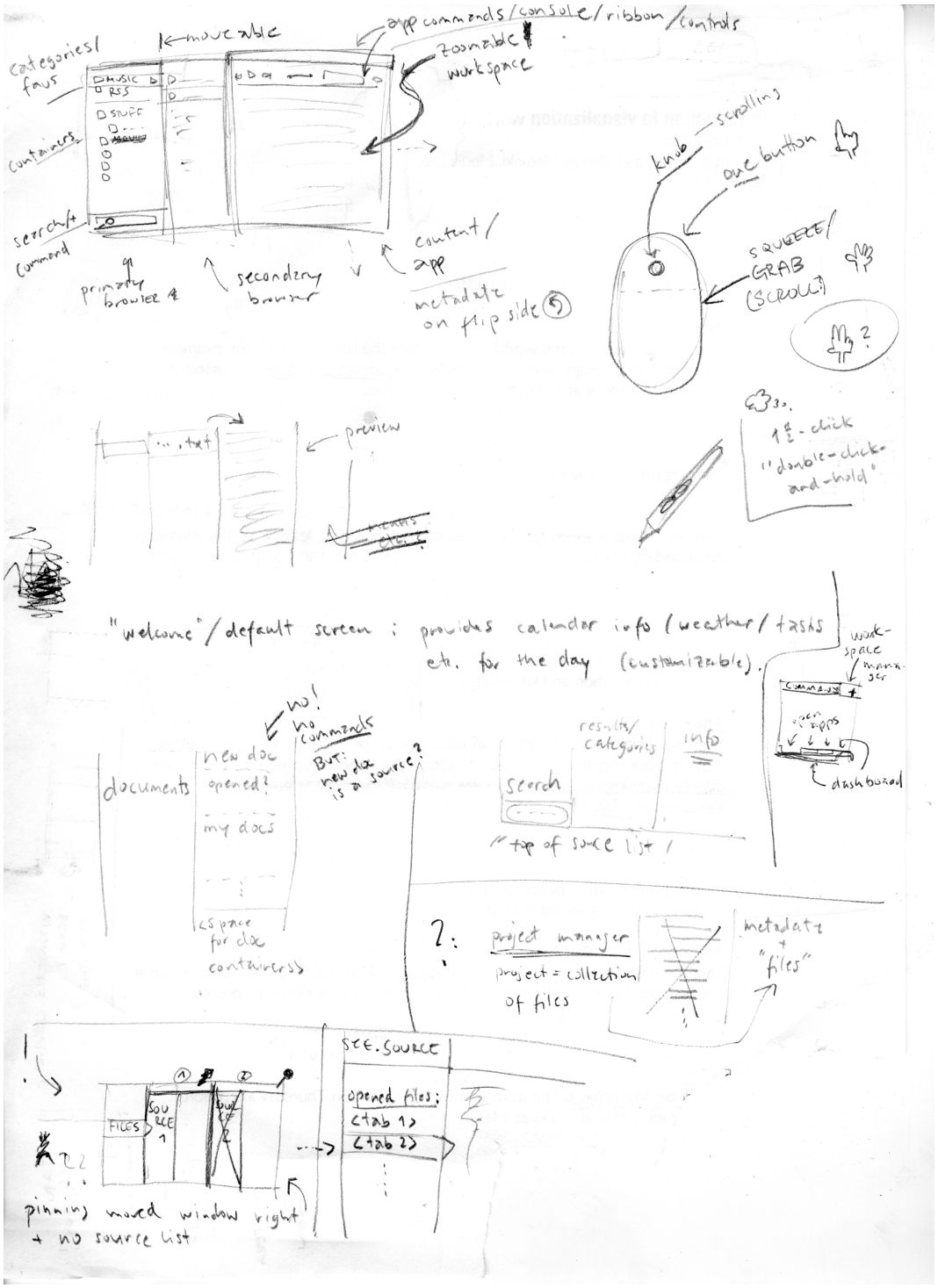
Figure 24 (a) Yahoo! Incorporated. Via: Ars Technica. http://media.arstechnica.com/reviews/apps/konfabulator.media/konspose.gif [accessed: October 5, 2008] (b) Gimpl, Martin. Screenshot of Mac OS X, courtesy of Apple Incorporated. October 5, 2008. (c) Microsoft Corporation. Via Techlog. http://www.techlog.org/images/vista_ctp3_sidebar.jpg [accessed: October 5, 2008]

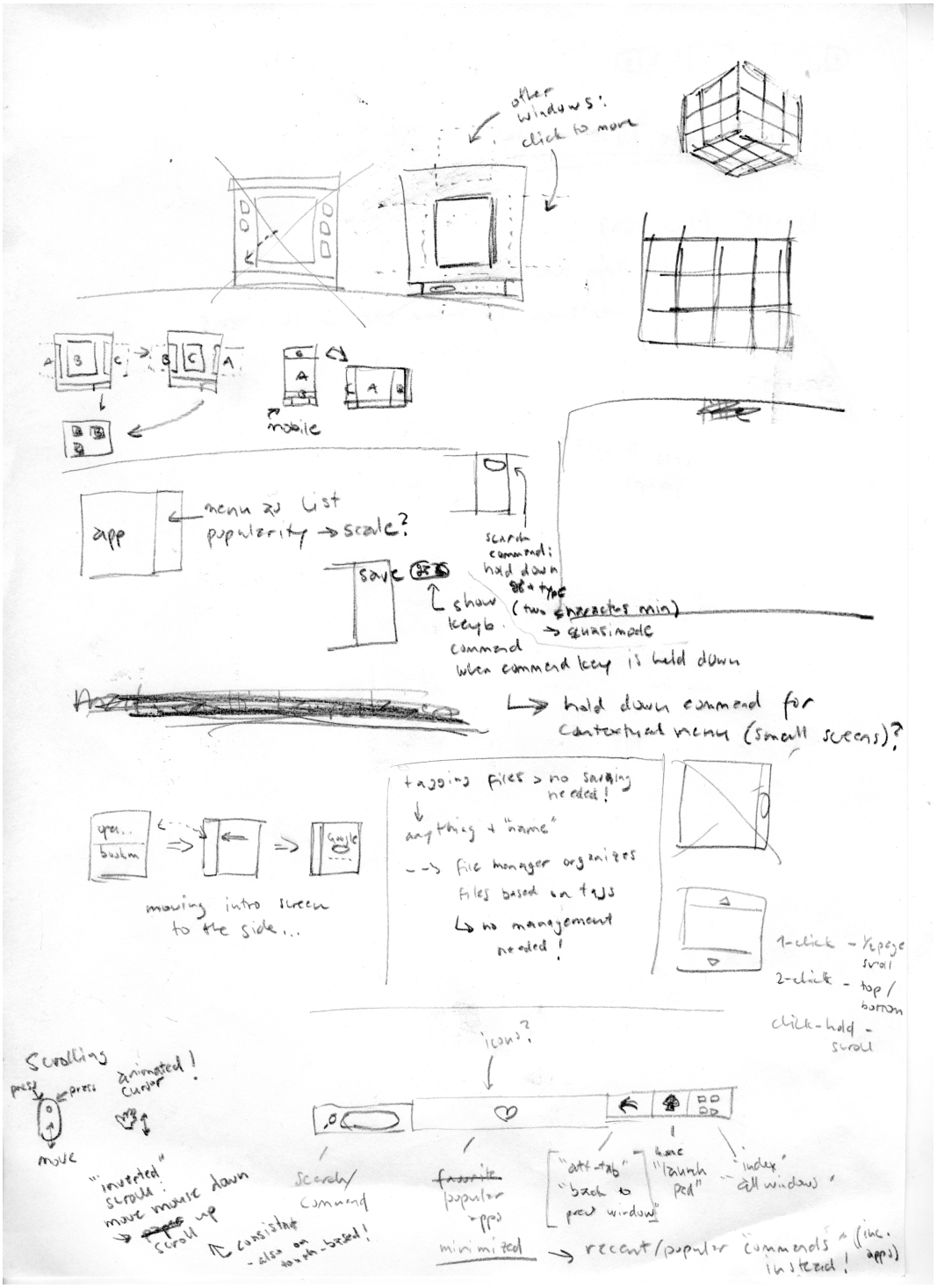
Figure 25 Microsoft Corporation. Composite image. Via: Software Usability Research Laboratory, Wichita State University. http://www.surl.org/usabilitynews/41/images/adapt_1.gif, http://www.surl.org/usabilitynews/41/images/adapt_2.gif, http://www.surl.org/usabilitynews/41/images/adapt_3.gif [accessed: October 5, 2008]

Figures 26–48 Gimpl, Martin. October 15, 2008.

Appendix

A Design mockups




other windows: click to move
mobile
app
menu w/ list
popularity → scale?
save
search command: hold down ⌘ + type (two characters min)
show keyb. → quasimode
command
when command key is held down
hold down command for contextual menu (small screens)?
open..
bookm
create
moving into screen to the side..
tagging files > no saving needed!
anything + "name"
--> file manager organizes files based on tags
no management needed!
1-click - ½ page scroll
2-click - top / bottom
click-hold - scroll
Scrolling
press
animated! cursor
move
"inverted" scroll: move mouse down
scroll up
consistent - also on touch-based!
scroll/ command
favorite
popular apps
minimized
"alt-tab" "back to prev window"
home "launch pad"
"index" "all windows"
"icon"?
recent/popular commands (inc. apps) instead!

B User test task list

Welcome, and thank you for taking the time to test Stripes, a conceptual operating system user interface. The test will be conducted on a rough prototype of the system, and aims to find out any major problems with its design. Keep in mind, what is being tested here is **not** you, but rather the concept.

Here's the backstory: you have just booted up your computer in order to complete the two tasks you had on your mind for today: **adding your favorite food to your grocery list** so you won't forget to buy it at the store, and **preparing a snippet of code** for your friend Steve, who has asked for your help. And while you're at it, you might as well **check for new emails**.

You have just bought a new computer, and it has a new operating system that you are not familiar with, called Stripes. Yesterday, Steve set up the computer for you and with his help, you remember creating two documents and browsing the web for a bit. But now, you can't remember anything about the darn thing. Let's see if it's any good at helping you accomplish your tasks!

The computer seems to have booted up. **Enter Stripes.**

1.
Since you are not familiar with this new operating system, you are free to play around with it for a few minutes before starting with the tasks. Bear in mind that not everything works perfectly and that not every feature is supported in this prototype.

When you are ready to start the test, **just say so.**

2.
See whether you have any new mail. Once you have found out, **please say the number of new email messages out loud.**

3.
No new mail, it seems. But wait! You just remember about a website another friend has been recommending to you for days; it's called Cute Overload, and since she's a dog lover you have a hunch that something quite adorable is waiting for you. **Visit www.cuteoverload.com** and check it out.

4.
Enough cuteness, it's time to get some work done! You remember you wanted to add something to your grocery list, a document you created yesterday. It looked like a page out of a notepad, yellowish. **Find and open the grocery list.**

5.
The list of groceries is clearly missing something! **Add your favorite food to the list.**

6.
Great, you won't be forgetting anything at the store, now! On to the next task: the code for Steve. You already have the code ready in another document. **Find and open the document containing the code.**

7.
Ok, the code is right here **between the marked lines**. You know Steve can be a bit of a nitpick, so you decide to copy the code into a new text document, and then convert it to plain text format -- you've heard that coders like that! But first, you need to **select the code snippet and copy it.**

8.
Now that you have copied the code, you need to **create a new text document**, and **paste the code into it**.

9.
Finally, you want to convert the whole document into **plain text** format.

10.
Phew, you're done! Steve's really going to love you for this once you send it to him later on. **Return to the original document and close it**.

11.
You notice there are a few windows open now, but you aren't sure how many exactly. Try to get a **complete view of all the opened windows simultaneously**, so that you can better understand where everything is.

12.
Rearrange the windows into a new order. Then **focus on one of them**.

13.
You realize you have accomplished the tasks you set out to do, even if this operating system is a strange new environment for you! You can continue to play around some more, or end the test. Thank you for participating!