*Bachelor Thesis*

Simon Lou Schüler

# Opinionated Expressive Mac-Assed

*Designing original software that feels at home on the Mac*

*Work can be fun*
*and fun can be useful.*

ANDY HERTZFELD, 2010

*Bachelor Thesis by*
Simon Lou Schüler

*Submitted to the Faculty of Design at the*
University of Applied Sciences Potsdam

*Supervised by*
Prof. Boris Müller
Frank Rausch

Berlin, December 2025

# Contents

# *Foreword*

I grew up with the glow of Aqua—translucent buttons that shimmered like drops of water, icons that felt alive, and interfaces that seemed to breathe. Over the years, that glow flattened into clean, quiet surfaces. Software became neater, faster, and somehow less Mac.

I want to rediscover what made that early software feel special—not just visually, but emotionally. It is about taste, care, and the small details that make digital tools feel like they were designed by people, for people.

CHAPTER 1

# *Introduction*

Building good software is hard—not because we lack tools, but because we often fail to learn from what came before. Over the past two decades, the platform Mac has moved through dramatic shifts in visual and interaction design: from richly textured skeuomorphism to todays cleaner, flatter aesthetics. Each era brought its own philosophy, strengths, and pitfalls.

Right now the Mac is in a weird place. On the one hand, it has been there for decades, developing a rich and joyful community around it. On the other hand, it is in a deeply fragile phase of self-discovery—currently very indecisive and unfinished feeling. The platform feels caught between philosophies—hesitating, shifting, and sometimes contradicting itself.

The whole software industry managed to navigate itself in an age of homogenization. As design systems matured and platform conventions solidified, software across all operating systems began to converge—visually, structurally, emotionally. Apps once had an individual identity, not feeling all the same. Their personalities faded, their quirks disappeared, and the sense of soul that once defined great software got lost along the way. Like Andy Allen (2020) puts it:

> *The world of apps—once an exciting canvas for creative exploration—has become repetitive, predictable, and… boring.*

'Rich Corinthian leather' originally referred to a made-up luxury car upholstery term from 1970s Chrysler ads, later used humorously for Apple's leather-textured skeuomorphic interfaces.

I want to open my eyes to the great things we had. It is not about bringing back *rich Corinthian leather* or intricate stitchings, but giving an app an identity, which often materializes in the details and the craft. It seems like many have forgotten that friendliness and fun are at the roots of the Mac. "Everything about the Mac has been trying to humanize technology." (BRAMHILL, 2015B, 9:08) From the 'Hello when the Mac first launches to the Finder icon that represents a smiling face.

The future of the Mac lies in synthesis—from photo-illustrative to flat design. By blending the warmth and clarity of earlier eras with the simplicity and focus of modern practice, we can imagine software that feels both contemporary and deeply human. It is okay for software to feature unique taste.

This thesis asks three questions:

1. What shapes the identity of Mac software?
2. How is that identity changing—and where might it go next?
3. How can designers create original, characterful work in a landscape where conventions are strong and sameness is so widespread?

By looking back at two decades of Mac software design, this work seeks to rediscover what once made apps feel alive—and to understand how those lessons can guide the platform through its current uncertainties. In a world where our devices can do almost anything, the question is no longer 'what can we do?' but 'what should we do?'

This thesis does not approach Mac software design through Human–Computer Interaction theory or usability metrics. Instead, it adopts a media-studies perspective that treats software interfaces as cultural artifacts—designed objects that communicate values, ideologies, and identities through visual language, metaphors, and interaction patterns. The Mac is understood not only as a tool but as a medium shaped by historical context, technological possibility, and aesthetic intention. The method is observational and comparative. By examining Mac OS releases, system applications, and selected third-party software across the past two decades, this work analyzes recurring patterns, dominant paradigms, and moments of transition in design direction.


**Figure 1**
Altair 8800





CHAPTER 2

# *A Little Bit of History*

To understand current software and why we landed where we are, it is important to have a look at the past. Many of the structures that still exist today were established during the early days of computing.

## 2.1 Rise of the Personal Computer

*From a CLI to a GUI*

In the 1970s, computers were in a transition phase—moving from large, expensive, specialized machines into affordable devices for homes, schools, and small businesses. This era began with the invention of the microprocessor in 1971 (LAWS, 2018) and the release of the Altair 8800 in 1974, which was widely regarded as the first personal computer. However, it lacked a screen and consisted solely of a variety of switches and lights. Consequently, it was unable to perform any truly useful tasks. Nevertheless, it remained an intriguing device for hobbyists. It marked the beginning of making computers accessible to a broader audience (GATES, 2025).

The concept of the personal computer spread rapidly with machines like the Apple II, the IBM PC, and the Commodore 64 in the late 70s and the early 80s. For many people, these marked their first contact with a computer. But still, computers were far from the wide-


**Figure 2**
Apple Lisa Advertisement (1983)

*Image from Apple (1983)*

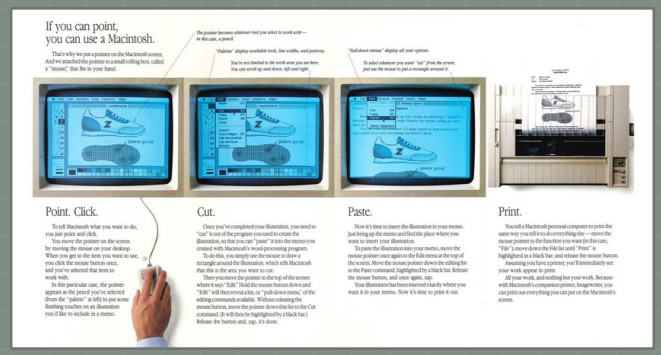



**Figure 3**
Apple Macintosh Advertisement (1984)

*Image from Apple (1984)*





spread distribution they have today (GATES, 2025). In the Current Population Survey in 1989, only 11% of computer owners said they bought their computer before 1984 (KOMINSKI, 1991).

Computers at that time were still controlled by command line interfaces. To appeal to a larger audience and make computing truly integrated into the lives of people, a new, simpler way of interacting with these machines was needed—according to Steve Jobs, a graphical user interface (GUI) controlled with a mouse was the answer (AS CITED IN ISAACSON, 2011). This way of interaction was mainly shaped by a research group at Xerox PARC. That is also where Jobs first encountered it in the context of a lab tour with the original Mac team (ISAACSON, 2011). In 1983, Apple first released a computer with a GUI in form of the Lisa and marked with that the start of modern computing as we know it today or in other words: "The Lisa contributed towards the idea that computers are for everyone." (ATKINSON, 2022A, 16:57)

With the Apple Lisa in 1983 and then the first Macintosh in 1984, the original Mac team around people like Bill Atkinson, Andy Hertzfeld, and Susan Kare, shaped the way we interact with computers. The introduced visual metaphors such as a desktop, folders, files, or a trash can made the computer and its behavior easier to understand—you could drag files into folders just like physical objects. These interactions and mental models largely remain the same until today (HERTZFELD, 2004). Alongside them, the use of icons to illustrate functions in user interfaces (UIs) is another first that has prevailed to this day (KARE, 1984).

## 2.2 The Story of Mac OS

*From gray boxes to translucent blobs*

In this brief overview, I will highlight the most significant design changes and phases in Mac software over the past four decades.

Both 'Mac OS' and 'macOS' refer to the operating system of the Mac. In this thesis, I will generally use 'Mac OS'. However, when discussing specific versions, I will switch between the two terms, as Apple renamed 'Mac OS X' to 'macOS' in 2016.

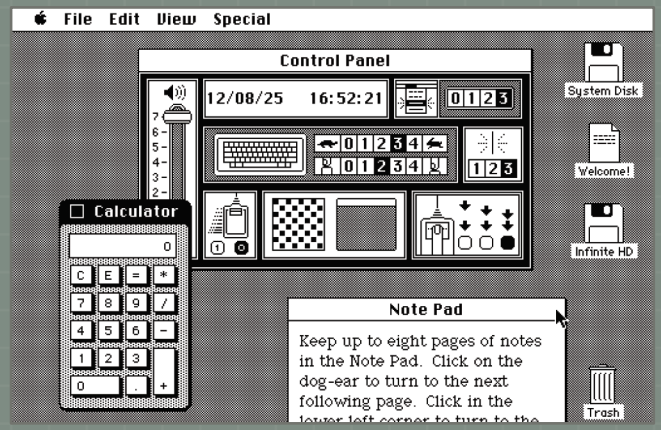



**Figure 4**
Mac OS System 1.0 (1984)


**Figure 5**
NeXT OpenStep 4.2 (1996)

*Image from Wichary (1996)*





## 2.2.1 Original Macintosh (1984)

*Birth of the GUI*

The first operating system (OS) for the Macintosh was based on the groundbreaking concept developed at Xerox PARC—a graphical user interface controlled with a mouse, which was considered very novel at the time (ISAACSON, 2011). Arguably, starting with System 1, the GUI of the original Macintosh shaped the way we are using a Mac until today. It introduced the the basic logic behind windows, menus and folders.

## 2.2.2 NeXTstep (1989)

*Foundation of the future*

Without being a Macintosh operating system, NeXTSTEP (the OS of NeXT workstation computers) is a noticeable part of the Mac OS design story. It served as the basis for building the first version of Mac OS X on top of it (MOGGRIDGE, 2007, PP. 141–142). It introduced several concepts that became integral to Mac OS X, with the Dock being the most prominent. Inspector panels, which give Mac windows a unique structure to this day, are also notable. Additionally, it laid technical foundations necessary for the following Aqua design language like translucency, real-time shadowing (MOGGRIDGE, 2007), and even core development concepts such as the AppKit framework. Its dominating medium gray tones, combined with black and dark blues, gave it a professional look that suited its customer base, which consisted of research institutions and professional developers rather than home users. The app and system icons were highly detailed for their time, showcasing a realism that was a stark contrast to earlier Macintosh versions (NEXT, 1995).

**Figure 6**
Mac OS X 10.4 Tiger (2005)

*Image from Hackett (2005)*

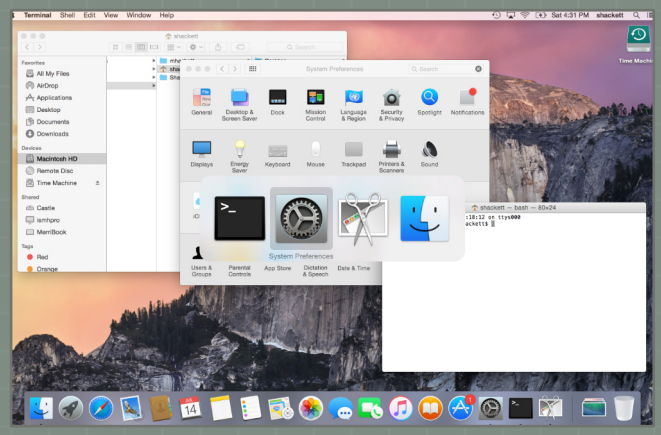


**Figure 7**
Mac OS X 10.10 Yosemite (2014)

*Image from Hackett (2014)*

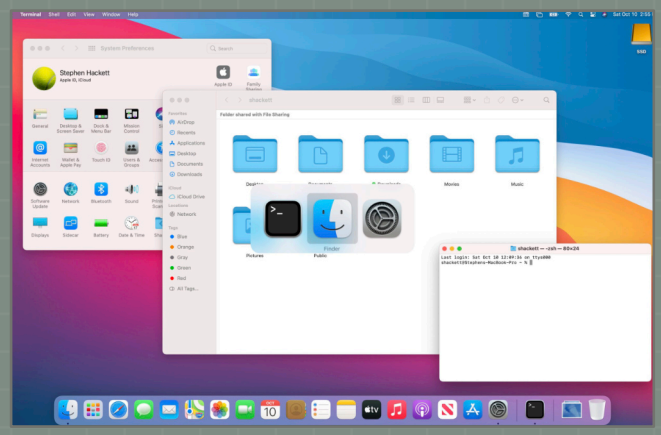


**Figure 8**
macOS 11 Big Sur (2020)

*Image from Hackett (2020)*





## 2.2.3 Aqua (2000)

*The era of emotion*

With OS X Cheetah (10.0), Apple introduced the colorful design language called Aqua featuring “things like translucent menus, gel-like buttons, and composite shadowing” (CORDELL, AS QUOTED IN MOGGRIDGE, 2007, P. 142), which where very novel and technically impressive for that time. Introducing Aqua, Steve Jobs said famously, “one of the design goals was that when you saw it, you wanted to lick it” (APPLE, 2000, 1:17:54). In addition to their visual depth, many apps faithfully replicated their real-world counterparts. For example, the calendar app displayed remnants of the torn-off paper sheet from the previous day under its toolbar. The use of these real-world metaphors made digital objects familiar to users new to the system (CRAMPTON SMITH, 2007, P. XV).

## 2.2.4 Flat Design (2013)

*Seeking clarity through reduction*

Starting with OS X Mavericks (10.9) and mainly developing with OS X Yosemite (10.10), Apple, under the design leadership of Jony Ive, transitioned to a design language that defines itself through reduction and simplicity. It clearly distances itself from the reliance on rich and detailed real-life references that prevailed before.

In the scene, this change of visual style was very controversial. Critics stated that it feels lifeless and in some areas less usable based on readability issues caused by missing contrast or shrunken drag areas (SIRACUSA, 2014).

## 2.2.5 Unification (2020)

*Decline of Mac identity*

In macOS Big Sur (11.0), Apple made a significant redesign to create a more consistent visual experience across their platforms, especially Mac OS and iOS. The

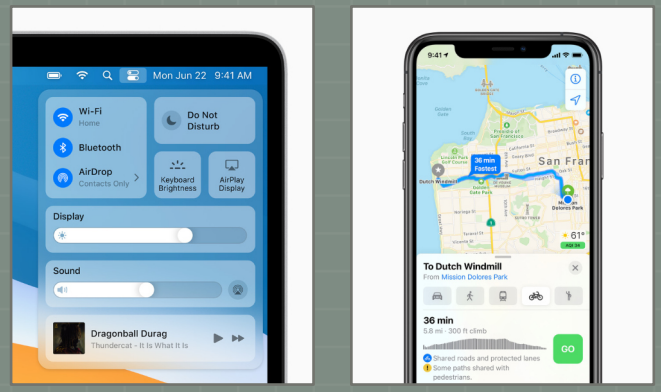



**Figure 9**
macOS 11 Big Sur compared with iOS 14

*Image from Apple (n.d.-a)*

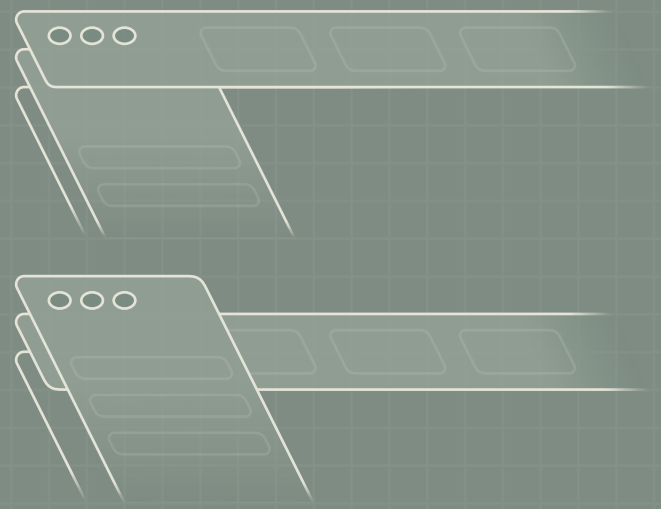



**Figure 10**
Comparison of toolbar placements





effort was to ensure continuity across Apple devices (APPLE, 2020). A notable example is the squircle shape of app icons, which was first introduced with the first version of iOS and now adapted for the Mac. In media coverage, this phenomenon is often referred to as iOS-ification (CUNNINGHAM, 2020). This often implies that design decisions are made in favor of iOS, and when viewed in isolation, they might not have been the optimal choices for the Mac—a trend that faced criticism for diverging from the core of Mac OS and for prioritizing style over usability (SNELL, 2020).

One of the main issues introduced in Big Sur is a hierarchical one, regarding the redesign of the sidebar, or source list, which now extends to the top-left corner of the window. Before, a toolbar on Mac stretched from the left to the right edge of the window, positioned at the top. With the logic that you read from the top-left to bottom-right, it clearly showed that toolbar controls can affect the whole window. This full-height placement positions the sidebar visually above the toolbar, creating a misleading hierarchical signal suggesting that the sidebar affects the toolbar, even though the toolbar generally remains constant as it is. A toolbar on Mac is intended to house frequently used commands of an app (APPLE, 2013A, P. 178). By elevating the sidebar above the toolbar, Big Sur disrupts the natural visual hierarchy between navigation and action elements. This marked the beginning of a shift away from a hierarchy defined by visual structure.

In addition to things like excessive padding around and spacing between elements, which can be attributed to the iOS-ification, Jason Snell (2020) pointed out that the curved edges at the top of menus are metaphorically incorrect. Combined with the previously discussed sidebar issue, this shows the willingness to compromise on hierarchical logic in favor of visual simplicity, marking a recurring theme I will further discuss in the next design phase.

**Figure 11**
macOS 26 Tahoe (2025)

*Image from Hackett (2025)*


**Figure 12**
macOS 26 Tahoe compared with iOS 26

*Image from Apple (n.d.-c)*





## 2.2.6 Liquid Glass (2025)

*Adaptivity over complexity*

With macOS Tahoe (26.0), Apple introduced a very broad update to their software design language alongside a second act of unification across their different platforms. The new design is much more expressive in itself through the newly introduced material, *Liquid Glass*. Alan Dye, the software design lead, describes it as "bringing joy and delight" and making it feel "more personal" (APPLE, 2025, 9:09).

Apple uses the term 'Liquid Glass' for their new design language, as well as for the material that is used for controls throughout the UI. Over the course of this thesis, I will therefore use that term for Apple's current design direction and for the material.

The design update has been widely criticized related to several issues, including legibility problems, unclear hierarchies, a mobile-first design approach, and an inconsistent implementation (HEER, 2025; VOORHEES, 2025).

A clear material like Liquid Glass, inherits the problem that content behind the element will affect the legibility of text on it. Especially with content that massively changes color, for example while scrolling. Apple is trying to mitigate this by switching the color of the control between a light and a dark version when the background changes, but the technical implementation lacks the quality to function reliably. A solution for this issue would be to make the material less transparent, thereby providing a more solid foundation for the text. However, this would inevitably reduce the visual impact of the Liquid Glass material and raise the question of whether such material is even necessary.

Structurally, the update changes the mental model of how Mac app windows are built, especially regarding the handling of toolbars, creating a massive impact on the perceived hierarchy. With the introduction of floating controls, toolbars as separate areas are removed. In the new paradigm, toolbar items just float on the content below and have no vertical separation to the content. This causes the problem that a control floating on content is automatically expected to refer directly to that content. However, according to Apple "a toolbar

gives users convenient access to the most frequently used commands and features in an app" (APPLE, 2013A, P. 178), meaning that toolbar controls can affect various different things and thus are not focused on the content below. It feels like Apple deprioritized visual hierarchy and focuses on contextual hierarchy and perceived simplicity in its new design. The designer and developer Nick Heer (2025) described it as follows:

> *Clarity and structure are sacrificed for the illusion of simplicity offered by a monochromatic haze of an interface.*

It is purely speculative why Apple did that, but perhaps an aspect of it comes down to the observation John Siracusa (2011) already made over a decade ago:

> *People will take "really cool-looking but slightly harder to use" over "usable but ugly" any day.*

## 2.3 UI Trends in Operating Systems

*Leaving room for expression*

Desktop operating systems are usually plain and restrained by design. They are made to be long-lasting, universally usable, and distraction-free, since people spend hours using them for focused work. A highly expressive or artistic interface—full of animation, texture, or bold color—might look exciting at first but could quickly become tiring, confusing, or inconsistent across millions of users and devices (WHITENTON, 2013). After all, the same system has to fit various different workflows with different requirements.

In the case of the Mac, Apple tried exactly that—keeping their system universal and simple enough to suit a lot of peoples needs without visual overload. To ensure those qualities, design decisions had to pass what Bill Atkinson called the "sanitation test" (ATKINSON, 2022B, 14:42) in order to get integrated into the OS. He wanted to implement flies that surround the trash can on the desktop when it is full—that did not pass the test.

Another aspect is that a visually reduced system appearance provides developers with more room to create expressive apps without clashing with the system. This isnt just a coincidence; it is an intentional design choice. In the early versions of Mac OS X, apps like Notes, Calendar, Reminder, and Contacts had very expressive interfaces. They balanced adopting the general structure of typical Mac apps with visually referencing their real-world counterparts. References being torn off calendar pages, the general layout of a phone book, or the typical yellow, lined notepaper. These combinations resulted in very expressive and unusual apps that still felt unmistakably Mac.

CHAPTER 3

# *The Ever-Changing Mac Identity*

The Mac always has had a strong identity. The definition of what feels at home on the Mac has a consistent core but is nonetheless constantly changing in the details depending on the current style.

## 3.1 Mac-Assedness

*Defining the term*

According to the working definition of the term 'Mac-assed', it refers to Mac applications that feel truly native. They are probably written in Swift or Objective-C for the best performance on Apple hardware and to be able to deeply integrate with the Mac OS system. Design-wise, they must adhere to the Human Interface Guidelines (APPLE, N.D.-B) for app navigation and structure to meet users' expectations regarding how to move through the app. In essence, these apps embrace the Mac's unique design principles, performance expectations, and user interface conventions, rather than being simple ports or cross-platform apps that do not feel integrated or natural on Mac OS. It is about software that does not just run on Mac, but thinks like the Mac.

What being Mac-assed does not mean to me is blindly following Apple's guidelines or design style. It is more

about serving the Mac as a platform that developed through the last four decades. What a typical navigation pattern is does not change with Apple's latest design update—they form themselves and harden over time and use. With the growing diversion of these long-developed patterns and conventions in newer Mac OS updates, it gets increasingly hard to maintain a design that serves the Mac and not only Apple's newest design style.

A common phrase among Mac developers is "being a good platform citizen" (MACDONALD, N.D.). It is important not to mistake that with being Mac-assed. People mean with that not only building a native app, but integrating as well with the system as possible—thus adapting Apple's current design language and utilizing its native controls. This often results in apps that look like apps designed by Apple. It also often includes the integration of the latest technical features and frameworks—being Mac-assed has nothing to do with that.

Important to mention is that 'Mac-assedness' is not to be equated with 'good usability', because Mac-assed describes the behavior that experienced Mac users would expect from their computer and not an interface that serves every user. There can be very Mac-assed apps that therefore integrate well with the system, but still are overly complicated and hard to use—especially for non-Mac users. After all, the Mac is a very opinionated system.

## 3.2 Exploring the Roots

*Where the Mac heritage comes from*

If you have been there, you probably find some joy in looking at Mac software from around 2010—people argue that the Mac was in its best state during that time. Surely a part of that is nostalgia, but I dare to say that there are other aspects to that as well. There are a few aspects I want to highlight that made systems like Mac OS X Tiger or Leopard great. Some are related to that time in technology, and others can be attributed to other values in the design process.

With the limited capabilities of web technologies at the time, most apps were native. Developers even embraced that, tailoring their apps to Mac OS and adopting Mac-specific patterns such as menus, sheets, and drawers. With most apps working fully offline, apps were judged on the quality of the software and not on their connectivity—it was not the norm that apps were available for all platforms.

People weren't afraid of complexity. While an app had to be usable and understandable, they were amazed by the capabilities of software. The joy of what software could accomplish outweighed the effort required to operate it. This made software often feature-rich and power-user-friendly, designed for productivity rather than minimalism—features were not sacrificed for simplicity. Doug Engelbart, who invented the computer mouse and pioneered early graphical user interfaces, once compared complex software with learning to ride a bicycle—it is a hard process that eventually unlocks power, which is worth it in the end (ATKINSON & HERTZFELD, 2010, 31:20).

The last point I want to make concerns craftsmanship—particularly on the Mac. There was a time when many developers approached their apps as passion projects, with less emphasis on immediate monetary value. Small teams, or even single creators, obsessed over details and stood out through design quality. A stark contrast to the current 'ship fast, iterate later' mentality and apps that are 'designed by committee'. A certain level of care, focus, and intentionality is often missing now.

A lot of the things I am going to address in this thesis were once predominant and got lost in technological change and the pursuit of continuous innovation. So we do not have to reinvent these things, we just have to rediscover them.

There are certainly many reasons why the scene has changed so much over the last decade—technological changes, the widespread adoption of software in everyday life, and, last but not least, Apple, which is currently focusing more on a unified ecosystem than on the Mac. There are people who say a bit cynically that all this quality between 2007 and 2011 comes down to the fact that higher management at Apple was busy inventing the iPhone (GRUBER, 2025)—but that's just speculation.

# 3.3 The DNA of the Mac

*What makes the Mac the Mac?*

## 3.3.1 Affordances of the System

The Mac system is built around a series of signature *affordances* (NORMAN, 2008) that shape the behavior users expect:

'Affordance' is a term to describe the inherent capability of an object or system—what actions it makes possible given a user's abilities.

1. **Selection and activation**
   Anything focusable can be selected; controls afford activation via click, tap, keyboard, VoiceOver rotor, and Spotlight. Double-click affords 'open', Return affords 'default action'.
2. **Manipulation of objects**
   Files, text, and UI components afford copy, move, rename, reorder, resize, and snap. Drag-and-drop between apps, spring-loaded folders, and text selection extend this.
3. **Navigation**
   Windows, spaces, and apps afford movement—Mission Control, App Switcher, Spotlight, and Finder's sidebar all let you traverse system state and content.
4. **Configuration and customization**
   System Settings, app preferences, and per-item inspectors provide the ability to permanently alter behavior. Furthermore, system-provided shortcuts for menu commands can be globally modified on the system level.
5. **Complexity and information density**
   The Mac affords high information density by enabling users to view and manage large amounts of information simultaneously through overlapping windows, precise pointer input, and UI elements designed for compact yet detailed displays. Together, these features support complex, multi-pane workflows and make it possible to work efficiently with dense, information-rich interfaces.
6. **Discovery and search**
   Spotlight, Help menu, and Finder search afford querying across apps and documents; Quick Look affords preview without opening.
7. **Feedback and undo**
   A systemwide undo/redo functionality allows safe exploration. Alerts, sheets, badges, progress indicators, and haptics confirm results and provide warnings before irreversible actions get executed.
8. **Window and layout control**
   Windows afford minimize, zoom, full screen, split view, and stage management. Panes afford resizing and arrangement.
9. **Security and permissions**
   The padlock, prompts, and sandbox afford granting/revoking access to files, camera, microphone, notifications.
10. **Consistent actions with shortcuts**
    Each app provides a set of basic actions that can be performed via a menu command or keyboard shortcut. For instance, '⌘ N' opens a new window or creates a new file, depending on the app, or '⌘ Comma' opens the app's settings in a separate window.
11. **Persistence and state management**
    Apps always auto-save the currently open document to prevent data loss and restore the state of the window or document after an app or system restart.

A 'signifier' is a perceivable indicator—visual, auditory, haptic, or contextual—that communicates what actions are possible or appropriate.

Many of the *signifiers* (NORMAN, 2008) of these affordances, in the form of visual clues, heavily changed over time or even disappeared. Buttons used to always have a carrier shape including a drop shadow, which made them clearly recognizable as an interactive control. The current selection of a picker in form of a segmented control in the toolbar, like the view selector in Finder windows, got much more reduced and lost contrast. This resulted in a loss of clarity, that was communicated through depths in older versions of Mac OS.

### 3.3.2 The Issue with Disappearing Signifiers

In a world where people are already familiar with a system and the metaphors used in it, it may be possible to reduce signifiers, as people can fall back on mechanics they have already learned. Nevertheless, when affordances lose signifiers, they become inherently harder to use until it becomes impossible, as an element can no longer communicate its purpose nor its behavior. As a result, new users in particular suffer when systems reduce signifiers, because this makes the learning process more difficult as elements become less self-explanatory.

For the sake of visual simplicity, a lot of systems remove signifiers and therefore remove visual cues of how you interact with it.

### 3.3.3 Human Interface Guidelines

Apple's Human Interface Guidelines are Apple's official design recommendations intended to help developers create apps that feel consistent across its platforms (APPLE, N.D.-B). They were once highly detailed and prescriptive, specifying clear rules for layout, component behavior, spacing, and interaction (APPLE, 2005). In recent years, however, many designers feel the guidelines have become less clear and less useful (HODSON, 2016). The process of developers to design an app that fits in well with the system is often not to reference the HIG, but to search for exemplary patterns in Apple's own apps (ARMENT & SMITH, 2025). App makers have even started writing their own HIG addendum to create a clear reference (GUZMAN, N.D.; RAUSCH, N.D.). That stands as testament to the current state of the HIG.

A major reason is that Apple has shifted from precise instructions to broad, conceptual principles. Instead of telling developers exactly how elements should behave, the guidelines now focus on high-level ideas like clarity or immersion, which are harder to apply in practice. With the current HIG being for all platforms at the same time, it is unclear which practice is aimed at which scenario on which device. Apple seems to focus on a perfect cross-platform experience and seems to have deprioritized good experiences on individual devices.

Internal inconsistency across Apple's apps further weakens the guidelines' authority. Navigation styles, toolbars, and modals differ significantly between system apps, making it unclear which patterns developers should follow. At the same time, the guidelines have adopted more marketing-style language—emphasizing emotional qualities over concrete rules—which reduces their practical value.

Because SwiftUI handles many decisions automatically, Apple provides fewer explicit specifications, expecting developers to rely on framework defaults (QUINLAN & HAJAS, 2025). Combined with incomplete coverage of newer platforms and UI patterns, the guidelines now serve more as a broad introduction to Apple's design philosophy than as the rigorous, detailed reference they once were.

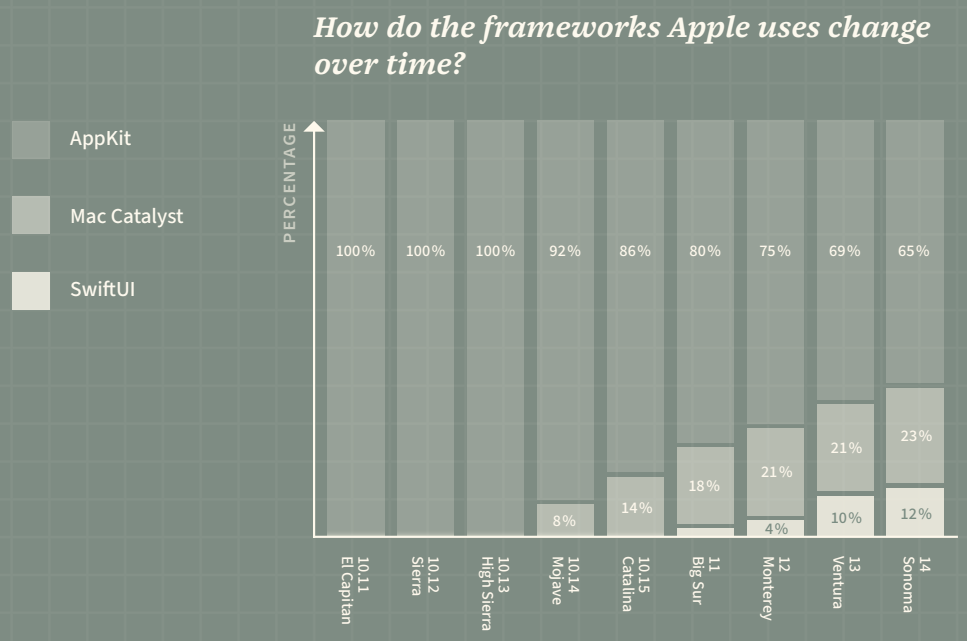



**Figure 13**
UI frameworks used in Mac OS since 2015

*Note: Figure created by the author using data from Colucci (2023).*





# 3.4 Technical Foundation

*Why technical decisions affect design direction*

One of the most crucial design decisions for a Mac app is its technical foundation, which includes the language and framework it is built upon. There are several frameworks by Apple, each suited to different needs.

In the Mac OS system, there's a noticeable trend indicating the growing adoption of Mac Catalyst and SwiftUI.

## AppKit

AppKit is the traditional, mature Mac OS framework—deeply flexible and historically rooted in Objective-C, though fully usable with Swift. With its roots in NEXTstep upon which Mac OS X was built, most of the system still relies on it (APPLE, 2013B). Though it was made clear in the WWDC 2022 State of the Union that this framework is not the future of development for the Mac (APPLE, 2022).

## SwiftUI

SwiftUI offers a modern, declarative way to build interfaces across all Apple platforms, and many Mac applications currently combine SwiftUI with AppKit to get both quick development and fine-grained Mac-specific control (COLUCCI, 2023). There are also several things you cannot achieve with SwiftUI on Mac—even basic things like creating a printing dialog—that's when people fall back on AppKit (SKOWRON, 2020).

## Mac Catalyst

Mac Catalyst provides another path, letting developers bring an existing iPad app to Mac OS while still adding desktop-centric enhancements, though Catalyst applications tend to have very simple layouts whose navigation patterns largely mirror those used on mobile. A common issue is that some behavior details of Catalyst

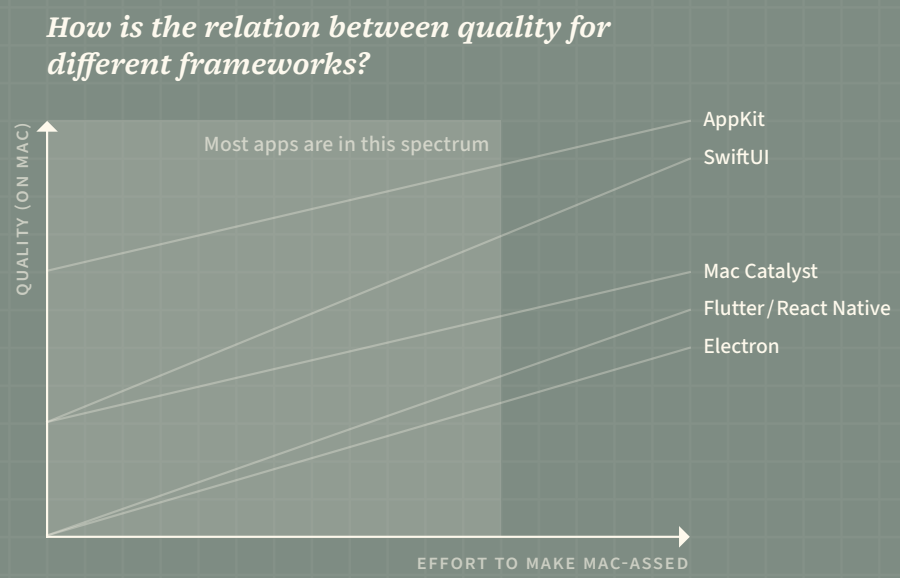



**Figure 14**
Relation between effort and quality for different frameworks

*Note: Since no empirical dataset exists for this relationship, the figure provides a conceptual visualization based on theoretical considerations and domain knowledge. It is not derived from measured data.*





applications work differently from those expected on Mac, which throws especially seasoned Mac users off (TROUGHTON-SMITH, 2022).

## 3.4.1 Effect on Mac-assedness

For comparing how different technical foundations influence the Mac-assed quality that applications can achieve, I propose an analytical framework. The framework is based on the idea that each development technology provides both an initial level of platform alignment and a certain potential for improvement through additional work. These two aspects together define a range within which applications built with the same technology can fall.

Different technologies start at different points. Some produce interfaces that align closely with Mac OS conventions right from the beginning, while others start much further away. At the same time, the extent to which developers can move an application closer toward a more polished Mac OS experience varies considerably. Certain frameworks offer deep access to platform features and afford fine-grained adjustments, whereas others impose architectural limitations that cap how far refinement can go, no matter how much effort is invested.

By describing each technology in terms of its starting point and its realistic ceiling, the analytical framework makes it possible to compare them not as fixed categories but as spans on a continuum. This allows for a more nuanced understanding of how technical foundations and developer effort interact. It also explains why applications built with the same framework can differ substantially: the technology sets the boundaries, but developers decide where within that range their application ultimately lands.

In summary, one can say that the chosen framework definitely influences the character of the software. However, this is not to say that AppKit software works better

in all cases, even though AppKit applications are more Mac-assed on average. It comes down to the amount of care or effort and ultimately to the level of dedication to build a Mac-assed app.

## 3.5 Software by Apple

*The most Mac-assed?*

Contrary to what one might expect, the Mac-assedness of Apple apps varies greatly. Depending on how long apps have been around, the extent to which they embrace the Mac identity varies. Older apps like Mail, Calendar, and Safari often have a more intricate window structure and adhere to typical Mac behavior, such as opening an email in a new window by double-clicking, showcasing their design for the Mac platform. Especially in more complex tools, above all Xcode, but also Final Cut X and Logic, you can see their long history and development—they use methods that are very specific to the Mac to make complex interfaces easy to use. In newer apps such as Music, Podcasts, and Passwords, the influence of iOS is very noticeable. They often have a very low-complexity, column-based layout, which works very similarly on mobile devices. These differences also reflect themselves in their technical foundation—older Apps are mostly built with AppKit, while newer apps rely on SwiftUI, or even Catalyst.

## 3.6 The Uncanny Valley of Mac Software

*When attempts at Mac-nativeness fail*

The 'uncanny valley' is the eerie, unsettling feeling people get when something looks almost—but not quite—human, such as a life-like robot or CGI face.

Although it is of course much easier to build a Mac-assed app with native frameworks such as AppKit or SwiftUI, there are other ways to do it. It is even possible with web technology, and attempts are being made to do so. The difference between these attempts and other cross-platform apps not specifically optimized for the Mac is hard to define. Maybe it is simply the effort involved in building a good experience specifically for the Mac (see *3.4.1 Effect on Mac-assedness*).

Still, in most cases, the feeling I am calling 'The Uncanny Valley of Mac Software' will remain—something feels off, different than in other apps. Overcoming this would require a great degree of attention and care for details.

This effect can come from a number of reasons. Cross-platform tools often lack full access to Mac OS system APIs, which prevents them from matching certain native behaviors. Their performance models—such as browser-based rendering—also differ from *Cocoa's*, creating small but noticeable mismatches in interaction and responsiveness. These limitations are reinforced by the inherent constraints of frameworks designed for portability, and by developers who may not be fully familiar with Mac OS conventions, leading to choices that subtly diverge from native expectations. To match the expectations of seasoned Mac users and give the app a Mac feeling, it needs sensitive decisions about when it is appropriate to use web technology and when to fall back on native frameworks.

Cocoa is the core set of native Mac OS application frameworks—primarily AppKit and Foundation—that shape app behavior, performance, and interaction patterns.

This phenomenon is not limited to web technologies; it also appears in Apple's own cross-platform approach with Mac Catalyst. Because Mac Catalyst adapts software originally designed for mobile, certain interaction patterns and behaviors carry over that do not fully match those of native Mac apps. Small differences in navigation, windowing, input handling, or layout conventions can make Mac Catalyst apps feel slightly out of place, reinforcing the same uncanny effect (TROUGHTON-SMITH, 2022).

## 3.7 Role of Fun

*Why the Mac has a special relation with playfulness*

The Mac platform has always had the philosophy of making everyday tasks a little bit delightful—from the original Mac's playful boot sounds to the way Mac OS animations feel organic rather than mechanical. Mac-native productivity software tends to embrace this by treating the interface itself as part of the creative process, not just a utilitarian shell around features. The whole development of the Mac was based on the goal of making it fun and easy: "We were trying to make the Mac for a 15-year-old. We were trying to make the Lisa for an office worker." (ATKINSON, 2022A, 23:58).

What's particularly interesting is how opinionated Mac software can be joyful just because it is opinionated. Having a very specific idea about how a respective task should be approached translates into interfaces that feel coherent and delightful to use. Not trying to be everything to everyone makes it easier to be the perfect tool for a very specific use case. This comes down to the dogma of 'quality over quantity'—building the perfect tool for a few people rather than a mediocre tool for many people.

Many aspects of the system became flatter and more neutral as it matured, but there are still remnants of that playfulness, such as preferences named things like 'Mission Control', that remind you of a time Apple took things a little less serious.

## 3.8 Recent Developments

*A liquid, glassy transformation*

In the following, I will shortly summarize the recent developments regarding Apple's software design language and move it into perspective to get a picture of the current state of the Mac.

In June 2025 at their yearly developer conference WWDC, in the context of macOS Tahoe, Apple introduced a very broad update to their digital design language, which will have a large impact on developers who are working on apps for their platforms. The new design language is called Liquid Glass. It introduces layered translucency across sidebars, toolbars, and inspector views. This shift emphasizes material, blur, and elevation over hard separators, relying more on type and spacing to establish hierarchy. It guides apps toward lighter chrome and contextual depth, relying more on dynamic contrast, which sometimes results in critical legibility. Widgets, panels, and floating palettes adopt the same glassy layering and large corner radii, promoting visual consistency across the system (APPLE, 2025).

Apple's idea behind this new visual language is to enhance the connection between software and hardware. For instance, the reuse of shapes like corner radii—buttons placed in corners reflect the curvature of the rounded display—demonstrates this concept. The playful animations should convey fun and lightness (APPLE, 2025). Alan Dye claims it "brings joy and delight" and feels "more personal" (APPLE, 2025, 9:09).

All these changes carry big implications for software development for the Mac. Software makers not only have to adapt, they have to find ways to deal with the new language in combination with the visual identity of their own app, which is in many cases not only an extension of the system but an entity of itself. What Alan Dye now says about feeling "more personal" (APPLE, 2025, 9:09) really is related to the Apple brand, which gains personality while independent apps have to lose some to fit in.

### 3.8.1 Hierarchy and Depth

Since the early 2000s, Mac OS moved from photo-illustrative design, which was heavily relying on visual metaphors, to the very flat design style of the last

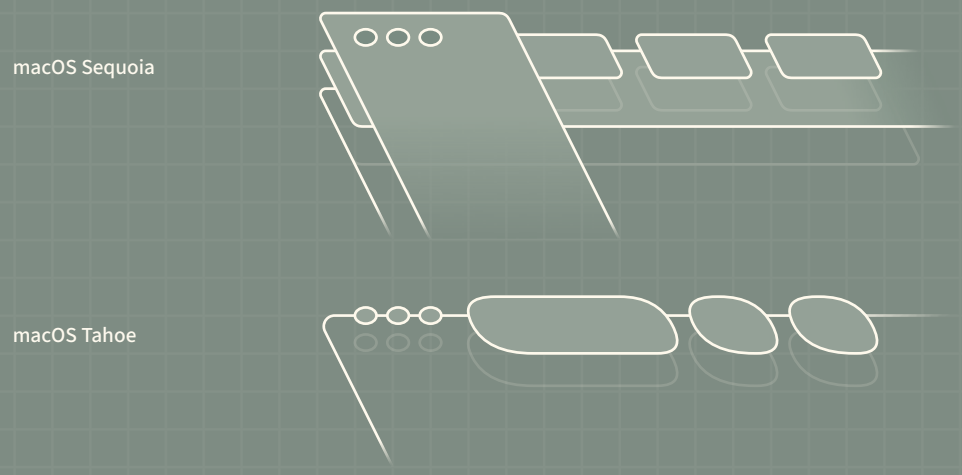



**Figure 15**
Comparison of the hierarchical layers in different Mac OS versions





decade. In some cases, there were major changes all at once, but much of it was also a gradual process.

For some time now, there have been signs that software design on Apple platforms would evolve in a slightly different direction—this has now resulted in Liquid Glass, a design style that reincorporates depth … or does it?

Depth can be separated into 'hierarchal depth' and 'sculptural depth'. The first describes the conceptual placement of an object to communicate its position in the navigational hierarchy. The second is just the visual or tactile impression of three-dimensionality in an UI element that has far less meaning except maybe providing a signifier that illustrates an object's function. In Apple's newly introduced Liquid Glass design system, you find conceptually only two layers in a window—a content layer and a control layer floating above. When speaking about hierarchical depth, you could say that the interface became even flatter, consisting of only two layers. Previously, a window had different areas that clearly had a hierarchical relationship to each other, which was more comprehensive and meaningful than just controls floating on a background. The sidebar was clearly above the toolbar, and the toolbar was clearly above the primary content area. Due to a very flat design language, this was less pronounced visually, but still clearly recognizable through subtle shadows and clear borders.

## 3.8.2 Apple's Abandonment of the Mac Identity

Beginning with the first wave of unification introduced in macOS Big Sur, Apple appears to be moving toward tighter visual and functional alignment across its platforms. Many of the decisions made during this transition seem to emphasize ecosystem-wide consistency, resulting in changes that differ from longstanding Mac conventions. Design decisions are being made in a manner that is detrimental to the Mac, in favor of the

overarching goal to present a more unified experience across its devices (SNELL, 2020). This trend only increases with macOS Tahoe and the corresponding Liquid Glass design direction.

Where does that leave the Mac? Based on what was discussed earlier in this chapter (see *3.3 The DNA of the Mac*), a quality decrease in some areas is noticeable.

The Mac is getting simpler, not in a usability sense, but visually. The window chrome is reduced, while buttons become larger and more rare. The perceived complexity is decreasing, while the actual complexity is not.

Although Apple has always made opinionated software, there was still freedom for configuration and customization on the Mac. Fewer apps now offer the option to customize the toolbar (see *5.1 Sketch*). Advanced settings are hidden deeper in the system settings.

To put these developments into context, it comes down to the defined purpose of the Mac, that very much is different from the purpose of an iPhone. Many people would classify the Mac as a tool for both complex professional tasks and everyday computing. Affordances such as 'configuration and customization' or 'complexity and information density' contribute to this.

### 3.8.3 A less-Mac Mac

*What if the Mac stops feeling like the Mac?*

Considering these developments, it feels reasonable to say that the Mac is becoming less typical Mac than it once was. This opens the room for the question of how the future of software design for the Mac will look for people who value the qualities of traditionally Mac-assed software. The question arises if it is even possible to create a truly Mac-assed app utilizing Liquid Glass and the newest changes regarding structure. In the process of unification (see *2.2.5 Unification*), Mac OS moved closer to iOS and adopted some of its both visual and navigational conventions (CHIN, 2022). To create an app that truly feels at home on the Mac, it may be necessary to compromise on the latest Apple style in favor of navigational conventions that have been the norm on Macs for decades. The examples analyzed later in this thesis demonstrate how that can look like (see *5 Looking at Examples*).

This is not really a new thing and can be viewed as a trend of recent years; even macOS Big Sur introduced quite a few things that "will throw long-time Mac users for a loop", as Jason Snell (2020) described it, but with macOS Tahoe the whole system made a significant jump in that direction. It leads to quite interesting questions when the prioritization of traditional conventions and navigations conflicts with Apple's new style, as it does, for example, in relation to toolbars (see *6.2.2 Building Toolbars*).

CHAPTER 4

# *Software as a Cultural Artifact*

Like other objects that are designed for use, software inherits not only the context of the time and social environment it was created in, but also reflects the creators' perspectives, assumptions, and values. In an era where software is so integral to our lives, it becomes a highly relevant cultural and social artifact. Yet despite its deep entanglement with contemporary life, software is still frequently approached in a purely utilitarian manner, reduced to functionality rather than understood as a carrier of meaning, ideology, and cultural influence.

As Andy Allen fittingly expressed:

> *It's just sad to me that there are very few people using software as a true medium for interesting ideas in the way that you see in industrial design, architecture, fashion, or even graphic design. I want to see software design sit up there on the cultural landscape at the same level.* (RIDDERING, 2024)

## 4.1 Visual Metaphors

*A translation from the real world*

Visual references have been fundamental in user interfaces since the beginning of personal computing, helping users understand digital concepts through familiar

real-world analogies. The most influential visual metaphor emerged at Xerox PARC in the 1970s with the desktop metaphor—documents, folders and a trash can (see *2.1 Rise of the Personal Computer*). The second big chapter to mention is the concept of skeuomorphism: digital objects resemble their physical counterparts. Especially early Mac OS X and iOS apps relied on these visual clues in the form of photo-illustrative design to provide cues about the underlying functionality.

Since the flat design era (see *2.2.4 Flat Design*), beginning around 2010 to 2013, many of these visual real-world references have been replaced with more straightforward interfaces. Many people argue that users do not need these metaphors anymore because people are used to interacting with software and therefore need less explanation of the mental model (EVANS, 2013). Furthermore, people may be more used to the digital version than its analog counterpart—everyone is using a calendar app, but only a few are still using a physical wall calendar to plan their lives (NICOL, 2013).

## 4.2 Age of Homogenization

*The process of systematization*

Based on the simple realization that if you do everything the same, it becomes boring, and if you change everything, it loses recognizability, Paula Scher (2017) addressed the core of the need for a design system.

A design system is a collection of fundamental principles, patterns, components, primitives, and even thoughts regarding tone of voice. It serves as a framework to manage design at scale by minimizing redundancy and establishing a shared language and visual consistency across various contexts (FESSENDEN, 2021). It cannot be too restrictive but has to guide in a way to create a sense of unity:

> *Most design systems I see today are just glorified component libraries.* (WORBOYS, 2024, 17:49)

As Perez-Cruz (2019) puts it:

> *Leaving room for experimentation is a tough balancing act.*

### 4.2.1 Design Language vs. Visual Language

The Design language is an overarching collection of principles, guidelines, and philosophies that shape how a system works, feels, and communicates. It is less about appearance and more about the structure, hierarchy, interaction patterns, and the logic behind how elements behave. For example, Apple's HIG provide a design language that defines how apps should be structured and how users should be able to interact with them. Famously described by Apple's former CEO and founder Steve Jobs:

> *It's not just what it looks like and feels like. Design is how it works."* (JOBS, AS QUOTED IN WALKER, 2003).

The Visual language is a subset of design language. It is built on elements like line, shape, negative space, typography, iconography, color, and texture. These are arranged according to principles such as unity, hierarchy, balance, contrast, scale, and dominance. While these elements and principles form the building blocks of visual language, the way they are applied is guided by the broader design language (IXDF, 2016). Following Apple's visual language is less essential for a Mac experience.

You can design an app that feels 'at home' on the Mac by following the HIG's design language—making it familiar and accessible—while still expressing your own visual style. This separation allows for originality, brand differentiation, and joyful details even within the same platform guidelines (IXDF, 2016).

In the broader context of this thesis, it means that it is possible to separate adapting Apple's design language from following their visual language.

### 4.2.2 Issues with Design Systems

Repetition and standardization are foundational concepts in the application of design patterns within software development. As outlined in recent literature, design patterns offer proven solutions to problems that recur throughout the design process (TIDWELL ET AL., 2020). By capturing expert knowledge and organizing it into reusable templates, design patterns enable developers to address similar challenges efficiently and consistently. This process of repetition ensures that effective strategies are not reinvented for each new project, while standardization provides a common language and structure that enhances collaboration, quality, and the overall coherence of software (ASGHAR ET AL., 2019).

> *[Design systems are a] powerful force for consistency, but don't support creative freedom and expression.*
> (LASH, AS QUOTED IN WORBOYS, 2024, 12:00)

While repetition and standardization streamline development and enhance reliability, they inadvertently lead to a subtle drawback: the gradual decline in diversity and originality in software design. As software makers increasingly rely on established patterns, the unique characteristics that once distinguished different software begin to fade. Over time, this widespread adoption of familiar solutions can result in software that feels homogeneous, predictable, and even uninspired. The very mechanisms that make development efficient and consistent may also contribute to a landscape where innovation is stifled and new ideas disappear, making software progressively more boring as it adheres to the same tried-and-true templates.

### 4.2.3 Expressiveness in Uniformisation

*Uniformity created by an expressive system*

Very expressive design systems are increasingly visible in digital product design, where color, shape, motion, and other visual elements are used to create stronger emotional responses and more memorable interfaces. For example, Google's Material 3 Expressive explores the use of bold shapes and colors, aiming to make key actions more noticeable and interfaces more appealing. Google researchers claim that sometimes expressive features can help users find important elements faster or make interfaces feel more modern and relevant (BENTLEY ET AL., N.D.).

Apple's Liquid Glass material, introduced as part of macOS Tahoe, is another example for very expressive and special elements packaged in a design system.

A challenge arises when expressive design systems blur the boundary between platform identity and the individuality of the apps running on it. Strong visual styles—such as bold color schemes, distinctive materials, or dynamic motion patterns—can become so dominant that apps struggle to maintain their own character. Developers may feel compelled to adopt expressive platform components even when these choices do not fully align with the app's tone or purpose. On the other hand, when apps try to push back and adjust or override expressive system elements to preserve their identity, they risk creating inconsistencies and breaking platform expectations. As expressive design systems grow more assertive, finding a balance between a unified platform feel and meaningful app-level differentiation becomes increasingly difficult.

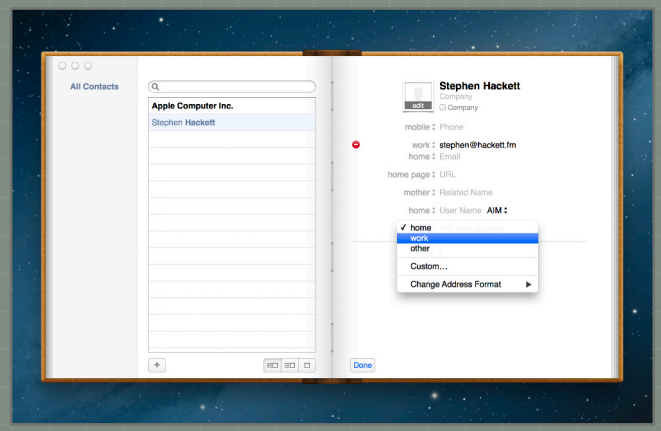



**Figure 16**
Contacts application in Mac OS X 10.8 Mountain Lion

*Image from Hackett (2012)*

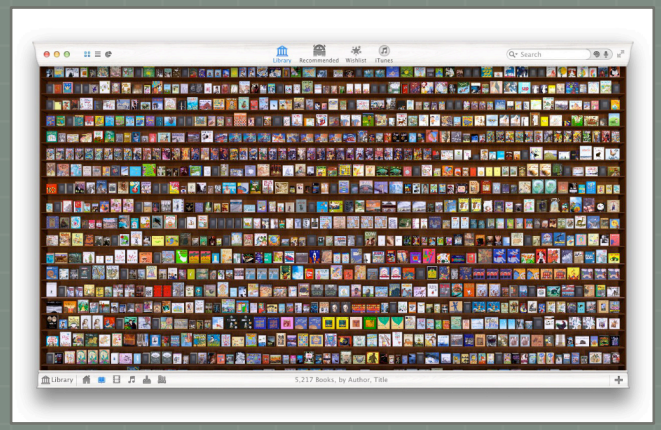


**Figure 17**
Delicious Library 3

*Image from Delicious Monster (n.d.)*





# 4.3 Expressive Software

*How apps can be different*

## 4.3.1 Room for Expression on Mac

In earlier stages of Apple's platform evolution, the system's design language—while distinctive—left meaningful space for individual interpretation. Developers could rely on Apple's visual framework for coherence while still shaping their applications with custom controls and unique aesthetic choices. The system set a tone, but it did not dictate the full expressive range of an app. This balance allowed the platform's developer community to flourish: indie teams could stand out, experimentation was common, and the overall ecosystem benefited from a diversity of visual and interaction approaches.

Apple itself developed a few apps that significantly deviated from their own guidelines for supporting a visual metaphor they desired to incorporate. Contacts, for instance, looked like a physical notebook. Apps like Delicious Library built on this way of interacting with defined guidelines and shaped how people would push the boundaries of the platform to build something unique and at the same time very well usable. The big difference from today's design style is that an app's visual language reflected its purpose.

In contrast, Apple's current design direction has shifted toward a far more expressive and unified aesthetic. With the introduction of context-driven hierarchy through fluid transitions between states with intricate motion behaviors, the system itself now carries a strong artistic identity. Rather than acting primarily as a structural foundation, it functions as a complete aesthetic environment. This elevates overall polish and creates a high consistency across the platform, but it also narrows the space in which apps can visually differentiate themselves without appearing stylistically out of place.

For larger companies building cross-platform products, this creates a practical challenge: replicating Apple's increasingly detailed system behaviors with non-native tools becomes difficult, if not impossible. Yet Apple has little incentive to accommodate such constraints, and might even advocate for building native apps like this. The more complex and distinctive the system aesthetic becomes, the stronger this implicit pressure grows.

The situation is more delicate for indie developers. The expressive richness of Apple's current design language raises the baseline visual quality of all apps, but at the same time makes it harder for small teams to carve out their own identity. When the system's personality is so strong, standing out can require rejecting core system conventions—an approach that risks confusing users or breaking the immersive cohesion Apple aims to maintain. This leaves developers in a tension between conformity and individuality: follow the system and risk sameness, or deviate and risk inconsistency.

This development leads to a broader and still unresolved question: if Apple's design language becomes so intricate and dominant that it limits meaningful self-expression, does the platform risk weakening the creative diversity that historically made it so vibrant?

## 4.3.2 Functional Expressiveness

Personality in software is not just aesthetic—it can enhance usability. Custom interfaces that reflect an app's core purpose (like Logic Pro's mixing board metaphors or Final Cut's magnetic timeline) can be both expressive and better usable than generic controls.

Garage Band in the 2010s had wooden bezels, which were unique to this specific app and immediately communicated which app you were in and what it did. An app can achieve the same effect when it is permanently in dark mode because it will stand out more when all other windows have a light appearance.

## 4.3.3 Playfulness

Playfulness represents a fundamental deviation from utility-focused design toward experiences that invite experimentation, discovery, and creative exploration. Rather than merely serving as efficient tools, playful software embraces ambiguity, encourages users to 'mess around', and treats interaction itself as a form of creative expression (KUTS, 2009).

This approach draws from game design principles while extending beyond entertainment into creative tools, educational platforms, and even productivity software. Playful interfaces might feature unexpected animations, hidden Easter eggs, or systems that respond to user input in delightfully unpredictable ways. They often prioritize emotional engagement over pure efficiency, creating moments of surprise and wonder that transform routine tasks into opportunities for creative play. Andy Allen (2024) demonstrates that a lot of ideas taken from the field of game design can be used to create a more varied and expressive software landscape.

In expressive software specifically, playfulness becomes a catalyst for creativity. Digital art tools that simulate the unpredictable behavior of physical media, music software that generates happy accidents through algorithmic variation, or writing applications that offer serendipitous word suggestions all exemplify how playful elements can unlock new forms of expression. These systems understand that creativity often emerges not from perfect control, but from the productive friction between intention and surprise.

The challenge lies in balancing playfulness with functionality—creating software that remains genuinely useful while fostering the kind of open-ended exploration that leads to creative breakthroughs. When successful, playful expressive software does not just help users create; it transforms them into active collaborators in an ongoing creative dialogue with their digital tools.


**Figure 18**
Neven Mrgan's button test

*Image from Mrgan (2013)*





# 4.4 Finding Style

*Let a button be a button*

A lot of designers struggled over the years with the issue of finding the right style. A number of contextual factors influence this challenge: the purpose of the app, the platform, and last but not least, the style that currently prevails in the industry—in summary, the context of the app.

Nobody would criticize when a book's design is trying to reflect its contents—meaning a cookbook is differently designed than a fantasy novel. For a good reason, I believe most people would agree that it's a completely reasonable thing to do. And so is it with literally everything. Of course, a resident home looks different than an office building—it has different needs and reflects its purpose. Why is it different with software? I believe it shouldn't be and therefore the app's purpose can be a primary source of design guidance for an app's interface. There are advantages to an accounting tool looking like an office and a journal app looking like leisure time. The aim here should not be to reference real objects in a skeuomorphic approach, but rather to reproduce a style or vibe.

What I also do not want to preach is that 'boring' things have to look boring. Just as you can make good use of a funky pen, the same applies to software. Making things look good so that people want to use them should be a good thing.

When talking about the influence of the current style that prevails in the scene, Neven Mrgan's button test from 12 years ago serves as a prominent example (MRGAN, 2013). In Apple's shift in the context of iOS 7 towards flat design (see *2.2.4 Flat Design*), he developed a matrix for various button designs. At the time, the upper left variant—just gray text without a border—was meant as a joke, but over the last decade there have been quite a few buttons that looked like this (BRAMHILL, 2015A). Apart from certain usability concerns regarding

the upper left variants, the button test primarily addresses the issue of the right level visual complexity, which changes significantly due to the current style. According to Louie Mantia, an app has to match the "level of visual quality" (GRUBER, 2025) of the system. What is meant is that the approach and the relationship to design must be consistent.

# 4.5 Finding Balance

*Identifying the sweet spot*

A great part of the software creation process is striking the balance between many different design directions. In the following, I want to illustrate some considerations in the design process and ways to evaluate which battles to fight.

## 4.5.1 Expression vs. Usability/Accessibility

Expression in software is not something that needs to be 'balanced against' usability; it is often one of the ways usability is achieved. Distinct shapes, deliberate motion, or a particular visual tone can make controls easier to recognize and interactions easier to predict (LORANGER, 2015). Expression becomes a problem only when it interferes with what users need to perceive quickly: what is actionable, what is important, and what will happen next.

The practical question is therefore not whether to be expressive, but where expression helps the interaction and where it distracts from it. Interfaces can support a surprising amount of aesthetic and conceptual individuality without harming accessibility—especially when expression reinforces structure or communicates purpose. Conversely, expressive choices that introduce ambiguity break intuitive mappings or hide essential cues tend to degrade the experience, no matter how original they are.

Good design works in this in-between space: expressive enough to create identity and meaning, while restrained enough to stay legible and inclusive. The goal is not to avoid expression, but to make every expressive element carry interaction value.

## 4.5.2 Opinion vs. Adoption

Deciding where to adopt existing concepts, both from the platform and generally established navigation structures, and where to develop a custom solution is a more complex consideration.

50 years of GUIs have not only taught us a lot about navigation patterns but also developed conventions. For a lot of UI and software development problems, a thought process and established solution already exist. Like platform standards, conventions are not a bad thing—they help users quickly find their way around new software (DOOSTI ET AL., 2018). The issue now is that conventions also play a major role in the process of software homogenization.

Considering all this, I believe a reasonable piece of advice is to avoid doing something different just for the sake of it. Instead, find new solutions for existing problems and use this opportunity to improve something and maybe express yourself in the process. However, I do not want to imply that questioning the status quo is unnecessary. Quite the opposite, I think it is crucial for creating software that surpasses the average by evaluating existing paradigms and challenging them.

## 4.5.3 Originality vs. Mac-assedness

Apart from the general UI design conventions mentioned before, there's also the issue of feeling native on the Mac (see *3.1 Mac-Assedness*). The balancing act is to combine personal uniqueness with the conventions that shape a typical Mac experience. While the last chapter has already investigated what creates the Mac DNA, I am

going to illustrate here what can be adjusted to make an app more original and which parts should be better not touched.

To start with, it helps to separate structural expectations from expressive opportunities. Structural expectations concern the elements that users implicitly rely on: window behavior, menu placement, standard keyboard shortcuts, and the overall information hierarchy. These are the foundations that give an app its "Mac-ness," the invisible framework that makes a user feel immediately oriented even before they understand the specifics of what your app does. Altering these too aggressively tends to create friction, because it undermines years of learned behavior.

Expressive opportunities, on the other hand, are found in the layers built atop that structure. These include the visual language, custom controls that extend rather than subvert standard components, and subtle interactions that convey personality without forcing users to relearn the basics. This is where apps can claim their identity—through motion, typography, iconography, spacing, or even the attitude projected by microcopy.

The key is not to avoid originality but to channel it. A Mac app can absolutely break conventions when it has a good reason, but breaking conventions should never be the first solution. Every deviation imposes a cognitive cost, and that cost must be paid back through a clearly improved experience. If a custom element is merely different rather than better, the user will feel the difference as noise rather than innovation.

## 4.5.4 Playfulness vs. Productivity

Especially on the Mac, playfulness always played a role, which does not mean that productivity did not—quite the opposite. The first program to create spreadsheets for a personal computer, Visicalc, ran on an Apple II, the predecessor to the original Macintosh.

At first glance it seems like playfulness and productivity are contradictory, but what if that just in not the case? Andy Hetzfeld, one of the most important members of the original Macintosh team, once put it like this in an interview:

> *And to this day I think that concept is at the very heart of Apple, that work can be fun and fun can be useful.* (ATKINSON & HERTZFELD, 2010, 15:32)

Another mentionable aspect is that skeuomorphism—an often really playful and illustrative way to communicate meaning—makes a piece of software understandable and usable and therefore more productive.

CHAPTER 5

# *Looking at Examples*

In this chapter, I will analyze a number of currently available apps. I will examine individual design decisions and evaluate how they affect the apps' identity and the integration with the Mac OS platform. How do they balance customness and nativeness? What elements are custom and what are Mac OS defaults? How does the use of custom controls or navigational patterns affect their feeling of familiarity? Through this analysis, I aim to identify components that are essential to an app's Mac-assed feeling and those that can be adapted to support a personal style.

Most of these apps would be widely considered 'indie' apps. While the line between indie and commercial is often blurred—especially in the Apple ecosystem—I define 'indie' not just by team size but by intent: a personal or niche vision that resists mainstream homogenization. Some apps, like iA Writer or Ivory, are commercially successful but retain a fiercely independent design ethos, situating them in the 'indie-spirited' category.

I will focus on the version of each app that was available in December 2025. If there was no redesign for macOS Tahoe at that time, I will evaluate it in the context of macOS Sequoia.

**Figure 19**
Sketch window (Version 2025.2 Barcelona)

*Image from Sketch (n.d.)*

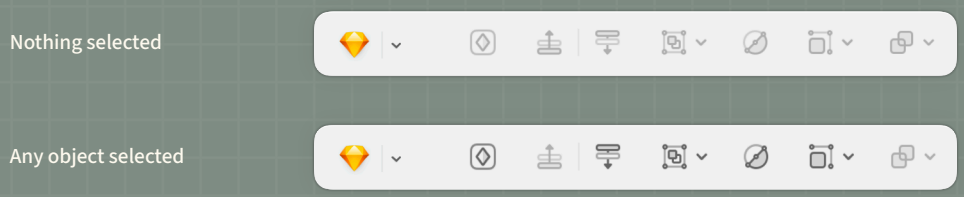



**Figure 20**
Constant toolbar
in Sketch (Version 2025.2 Barcelona) on macOS Sequoia


**Figure 21**
Five example states of the variable toolbar
in Sketch (Version 2025.3 Copenhagen) on macOS Tahoe





# 5.1 Sketch

*Barcelona (2025.2.4) + Copenhagen (2025.3.2)*

Version ‘Barcelona’ was developed for macOS Sequoia, and ‘Copenhagen’ for macOS Tahoe. Because the versions differ in some aspects dramatically, I analyzed them both and illustrated the changes.

Sketch is a canvas-based vector design tool for Mac OS focused on UI and product work. It organizes projects with artboards, reusable components, and shared styles. It also supports working locally, real-time collaboration, and interactive prototyping.

## 5.1.1 Structure

Sketch is organized into files—one Sketch window contains one open Sketch file, which can consist of multiple pages. Structurally, the app is organized around a large canvas in the middle, where the designed work lives. The left sidebar is vertically split and houses the pages of the current document and the layers of the current page. The inspector panel on the right contains all options and settings to manipulate color, shape, and behavior of the currently selected object. The toolbar gives access to the most important actions, which are also available via the app’s menus, as users would expect on Mac OS.

As the use and behavior of toolbars has undergone significant changes with the introduction of macOS Tahoe, Sketch also massively changed the behavior and aesthetic of its toolbar. As described above, the toolbar provided a constant set of the most important actions. From version ‘Copenhagen’, a variable toolbar has taken its place, which provides appropriate actions depending on the current state—object or tool selection. When viewed in the context of the Mac, this represents a drastic deviation from the established behavior of one of the platform’s most important patterns that users expect.

Apart from the convention break, there are upsides and downsides to this new behavior. The positive side is that functions are less buried in menus and dropdowns, as they appear when you could use them. The same applies to the selection of tools, as these are now dis-

played in the toolbar when no object is selected. I am critical of aspects relating to the discoverability of functions, because a possible action is only shown when the app is in a state where the former can be performed. If this state is not reached, you will not even find out that this function exists. In a traditional toolbar, as before, if an action was not possible at the current time, it was grayed out, which conveyed to the user that this function existed, but that they still had to do something to use it.

## 5.1.2 Behavior

While not all controls are native, all of them behave practically natively. Controls like shadows, colors, and effects have right-click actions to copy and paste them—just like you would expect on a Mac.

A noticeable unusual element is the tool selector, especially in version 'Barcelona', which is hidden behind the dropdown next to the Sketch logo in the toolbar. Tools are needed mainly to create new objects—shapes, frames, text, or drawings. At first glance, this seems very hidden for some of the most important functions. This could be because these functions are so essential that almost all users will access them via their keyboard shortcuts—after all, Sketch is a pro tool. Prioritizing functions or mechanisms for power users benefits the long-term efficiency here, as it saves valuable space. A typical solution to the tool selection would be some kind of always visible bar or panel like in Adobe Illustrator or Figma. The makers of Sketch decided here against the established solution. This makes it also a prime example for the topic discussed in the last chapter under the title '4.5.2 Opinion vs. Adoption'.

## 5.1.3 Visual Style

In terms of design, Sketch is not particularly expressive, nor does it have a distinct identity of its own. While it feels native, small details reveal unique custom solutions, showcasing a level of quality and craftsmanship.

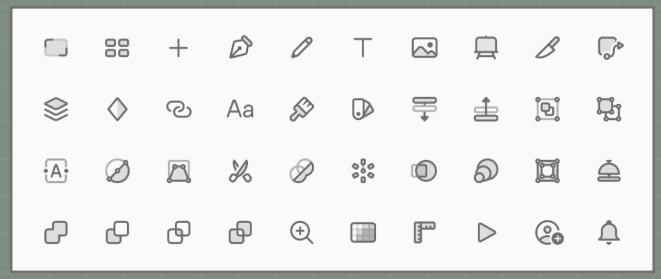


**Figure 22**
Sketch icon set from the 2020 redesign

*Image from Sketch (2021)*

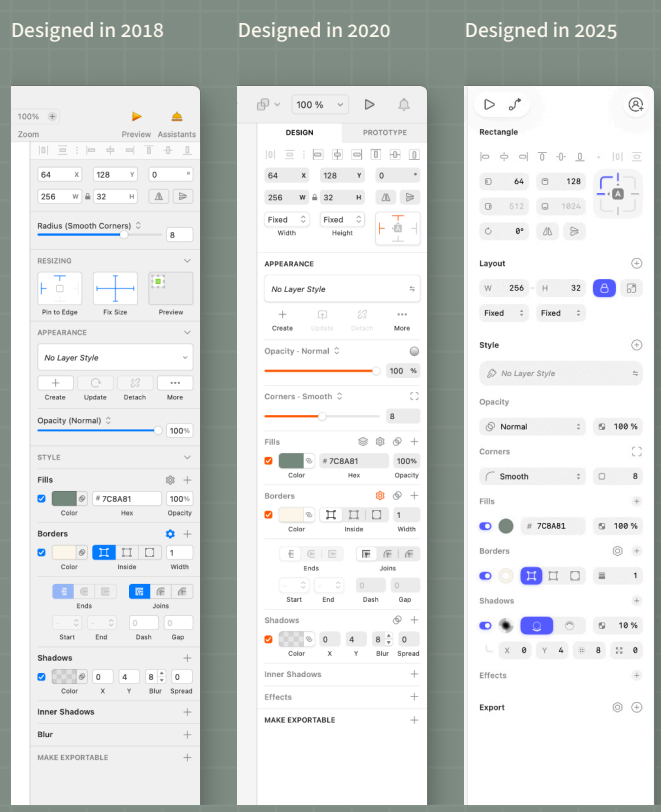



**Figure 23**
Sketch inspector comparison





Sketch's quality is demonstrated through its custom iconography. All icons in the app are meticulously crafted and designed specifically for the application. When introducing their new icon set, which remained in use until version 'Barcelona', they explained on their blog that the custom icon set was necessary due to the intricate meanings associated with individual symbols. Before, colored icons were used to communicate even more clearly, they only replaced them in favor of fitting in well with the system (SKETCH, 2021A). The new set is in a semi-filled style, meaning mostly outline icons with an additional gray fill to emphasize certain parts of the symbol. This is not only a middle ground between an outline and a filled style, but the sweet spot between the aesthetic of the system and an own unique style.

Because of the massive changes between the two versions discussed, it is necessary to focus also on the Inspector. Before, the overall look was in typical Mac fashion, very dense and somewhat crowded-looking. The updated version is more spacious and reduced-looking. At first glance, it seems like the new version does not significantly take up more space, but upon closer consideration, it becomes evident that multiple controls are either gone, only shown contextually, or hidden behind a second click. Elements like the sliders to control opacity or corner radius are gone completely, leaving only the option to set the values via an input field. With a few exceptions, the controls are still very usable. One exception is the button for displaying the prototyping options. Previously, the entire Inspector sidebar was divided into two tabs, 'Design' and 'Prototype'. Now, the latter is hidden behind a toggle button that just looks like a normal push button.

## 5.1.4 Settings

The app settings use a tabbed navigation typical for Mac apps. They use mainly native controls with some custom elements for uncommon interactions. They feel like a Mac user would expect them to.

## 5.1.5 Technical Foundation

Sketch for Mac is a native Mac OS application built with Swift, AppKit, and CoreGraphics. Its .sketch files use a JSON-based bundle format that stores documents as a hierarchical scene graph of pages, layers, and styles (SKETCH, 2022). The app loads and edits this structured model directly, and a patch-based system is used to synchronize document changes with Sketch's cloud services while still supporting local offline work (SKETCH, 2021B).

It is also possible to view the design files in a web UI. For that, Sketch's backend serves the document's JSON-based structure, which the web client loads and renders using its own JavaScript-based canvas engine (SKETCH, 2022).

This shows how the local app focuses on native frameworks for performance and offline capabilities while structuring data in a way that it can still be displayed in a browser.

## 5.1.6 Conclusion

In total, we see here an app that not only works really well on a Mac, but has a feeling to it that materializes itself through performance, quality, and well thought-out experience.

We also observe the application of context-based hierarchy, prioritizing the reduction of visual noise through hiding controls behind a secondary click or only revealing them contextually.

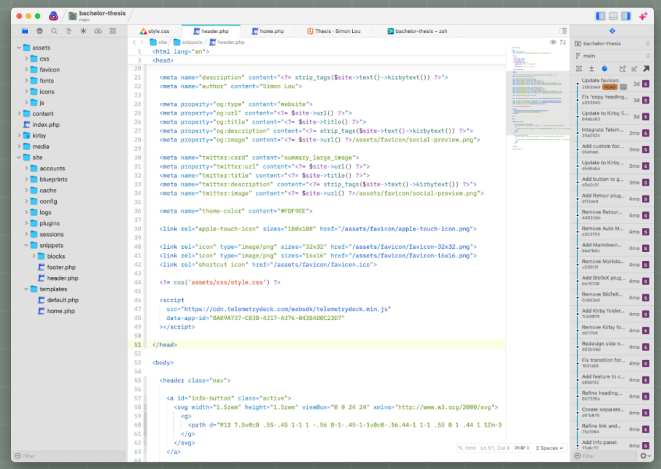


**Figure 24**
Nova window

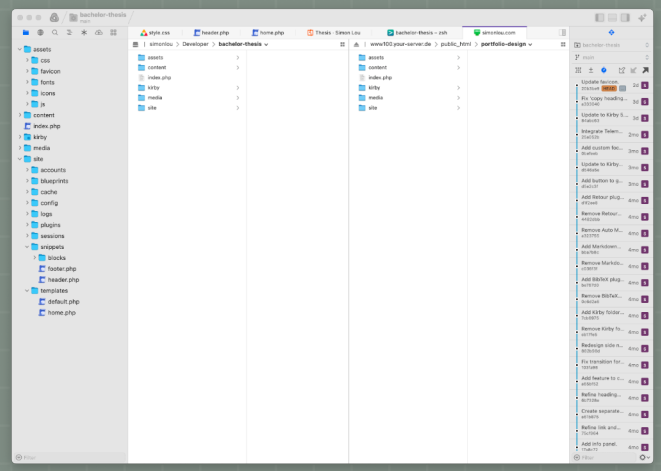


**Figure 25**
Nova file browser tab

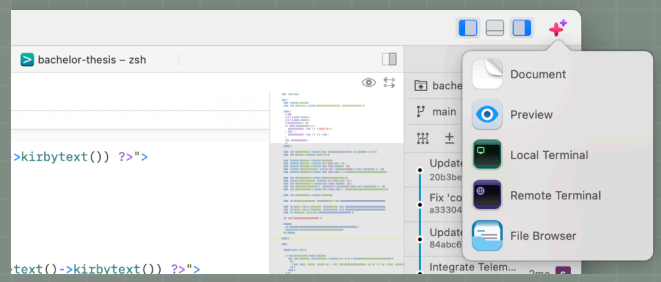



**Figure 26**
Nova tab types





# 5.2 Nova

*Version 13.3*

Nova is a code editor and development environment built for Mac OS, combining established development tools with an opinionated interface and a focus on extensibility. Its purpose is to offer a fast, native alternative to cross-platform editors while embracing workflows familiar to Mac developers.

## 5.2.1 Structure

Nova is structured around a tabbed, single-window workflow. Although file tabs were not part of the typical Mac OS patterns until 2016 (CUNNINGHAM, 2016), they have become very common in recent years—now found in almost every file-based app. Nova's smooth implementation makes them feel at home. The app also supports multiple tab types, such as documents, a file browser, an integrated terminal, or a web view, which is very unusual on Mac OS but useful in this case, as the development process demands different tools at different stages in the process. Normally, all tabs in one app belong to the same category—websites in Safari, text documents in Pages, or spreadsheets in Numbers.

In addition to these tabs, the split view functionality allows for simultaneously looking at two related documents or even a document and a web view preview. It strengthens the feel that this is a mature development environment, rather than just a code editor.

## 5.2.2 Behavior

Nova behaves very much like a typical Mac app. Its source list follows platform conventions, even if the icons themselves are custom. Standard interactions—such as right-clicking to access Services—work exactly as expected. The interactive document path in the window title is another distinctly Mac behavior that reinforces the native feel. The only notable departure is

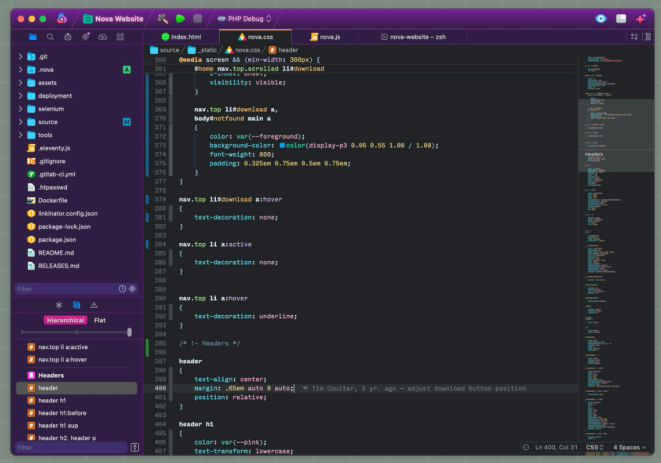


**Figure 27**
Nova with a theme

*Image from Panic (n.d.)*

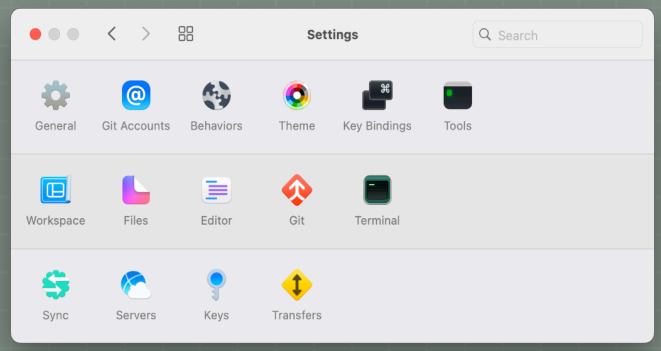



**Figure 28**
Nova settings





the non-customizable toolbar, which goes against common Mac OS conventions and limits user flexibility.

## 5.2.3 Visual Style

Visually, Nova mixes a strong custom identity with a few decidedly un-Mac touches. The launcher shown on startup—with its single close button—feels more like a bespoke panel than a standard system window. Likewise, placing the app icon in the toolbar is unconventional for Mac OS.

Nova's extensive use of custom icons gives the app a cohesive and memorable personality. Even elements that would have an equally fitting system equivalent are custom—for example, the source list icons like folders and documents. An interesting observation is that the app still feels very Mac-like. My theory is that this would not be like this with any other custom icon set—suggesting that breaking system defaults can be fine when doing it in a way that understands the platform.

## 5.2.4 Settings

The preferences window is modeled after the classic Mac OS design, using large, iconic 48px icons to represent categories, arranged in a grid. Inside each section, controls follow standard Mac interface components—checkboxes, pickers, and tab views—resulting in a settings experience that is both nostalgic and highly functional. The execution feels polished and thoughtfully designed.

## 5.2.5 Technical Foundation

Nova is a native Mac OS app, likely built using AppKit. Its responsiveness and adherence to many system behaviors reflect a mature and deeply integrated technical foundation. It shows how native development can enhance the user's experience in terms of quality and

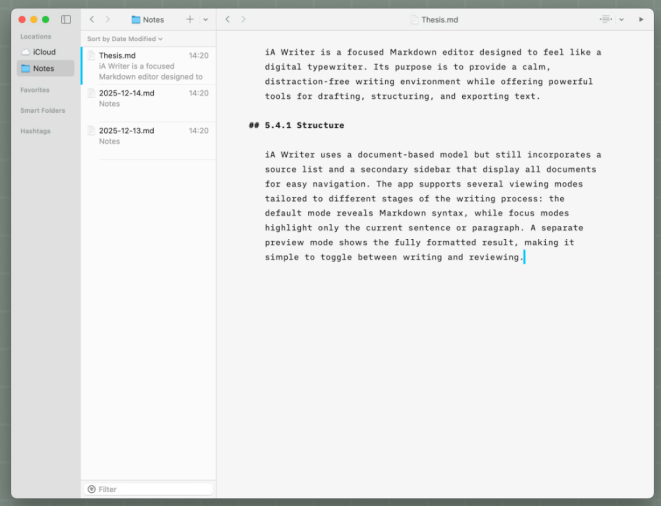



**Figure 29**
iA Writer window with shown source list





consistency, even in a field dominated by cross-platform apps like Visual Studio Code.

Extensions are an essential part of the Nova experience, as they expand the app's capabilities to complement a user's individual workflow. As they should be easily developable by many users, extensions are written in JavaScript (PANIC, 2025). This shows that native code is not always the best choice, as it would result in a less flourishing developer community here, because JavaScript is much more widespread.

### 5.2.6 Conclusion

Nova is highly custom in appearance, yet manages to feel entirely at home on the Mac. Its distinct visual language does not hinder familiarity; instead, it showcases how far stylistic individuality can be pushed while still maintaining a fundamentally Mac-assed experience. It is a prime example of how software can be very different and at home on a platform at the same time.

## 5.3 iA Writer

iA Writer is a focused *Markdown* editor designed to feel like a digital typewriter. Its purpose is to provide a calm, distraction-free writing environment while offering powerful tools for drafting, structuring, formating, and exporting text.

Markdown is a simple, human-readable markup language for formatting plain text that can be easily converted to HTML.

### 5.3.1 Structure

iA Writer uses a document-based model but still incorporates a source list and a secondary sidebar that displays all documents for easy navigation. The app supports several viewing modes tailored to different stages of the writing process: the default mode reveals Markdown syntax, while focus modes highlight only the current sentence or paragraph. A separate preview mode


**Figure 30**
iA Writer window in writing mode





shows the fully formatted result, making it simple to toggle between writing and reviewing.

## 5.3.2 Behavior

The app behaves very natively and places the writing experience at the center of its interactions. Features like sentence-level focus modes reinforce this intention, subtly guiding the user toward immersion. Everything responds in a familiar, Mac-appropriate way, keeping the experience frictionless.

Interesting to discuss is the decision not to adhere to the ‘what you see is what you get’ concept, like most writing apps do. For example, when using a tool like Pages, the format you see on the page is always what will get exported. iA Writer shows an abstraction from what will be exported—in relation to text format this will be both the format itself and the Markdown syntax.

## 5.3.3 Visual Style

Visually, iA Writer is simple and intentionally reduced—a kind of digital Zen environment. All contextually unnecessary controls are eliminated without compromising—arguably even enhancing—the app’s usability. For the remaining controls, iA Writer relies almost exclusively on system controls. This purposefully-minimal aesthetic supports the idea of getting everything unnecessary out of the writer’s way.

Another example of the dedication to a perfect writing experience is the app’s use of three custom typefaces. As described on iA’s blog: “… you probably won’t see any of this. But you might feel it.” (IA, 2018) Also, fonts for writing and reading, exporting, are not considered the same here. The iA Writer font reduces confusion and, being monospaced, creates a predictable flow. The export of a text can use all sorts of typefaces, defined by the selected preset.

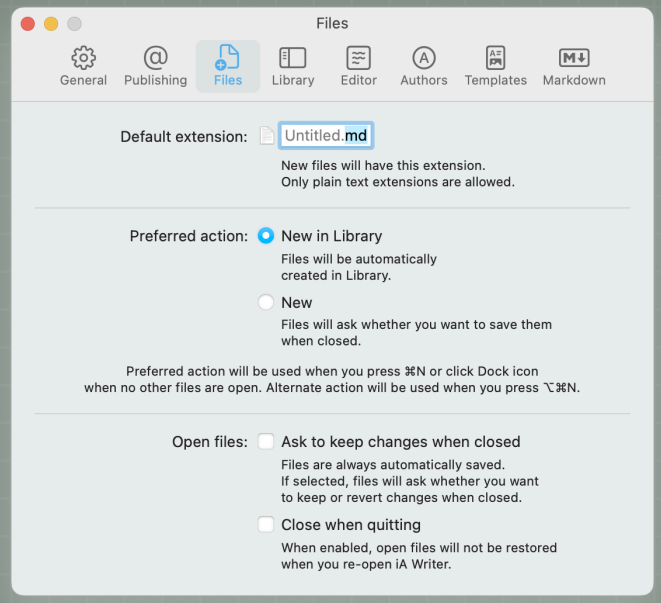



**Figure 31**
iA Writer settings





Even though its only deviation from system controls is its use of custom fonts, the app still feels very unique in its design language.

## 5.3.4 Settings

The preferences are straightforward and follow typical Mac OS conventions—placed in their own small window and organized through tabs. They feel clearly structured and offer the necessary options without introducing anything unusual or distracting. The use of checkboxes, rather than toggles, feels very Mac-like.

## 5.3.5 Technical Foundation

iA Writer is a native AppKit application. It works directly with Markdown files, reinforcing an open, portable workflow that avoids lock-in and keeps documents maximally flexible. Although the app displays and stores files in a library folder by default, it allows plaintext files from other locations to be opened, which makes it very flexible.

Another enabling feature is that the style in which documents are exported can be specified using a CSS stylesheet, allowing people with a simple understanding of web development to create their own layout with minimal restrictions. This enables people familiar with web technologies to produce finished documents while editing in Markdown.

## 5.3.6 Conclusion

iA Writer is a strong example of how expressiveness on Mac OS does not require bright colors or elaborate visuals. Its identity emerges through reduction and clarity, showing how a restrained design can create an experience that feels both distinctive and deeply Mac-like.

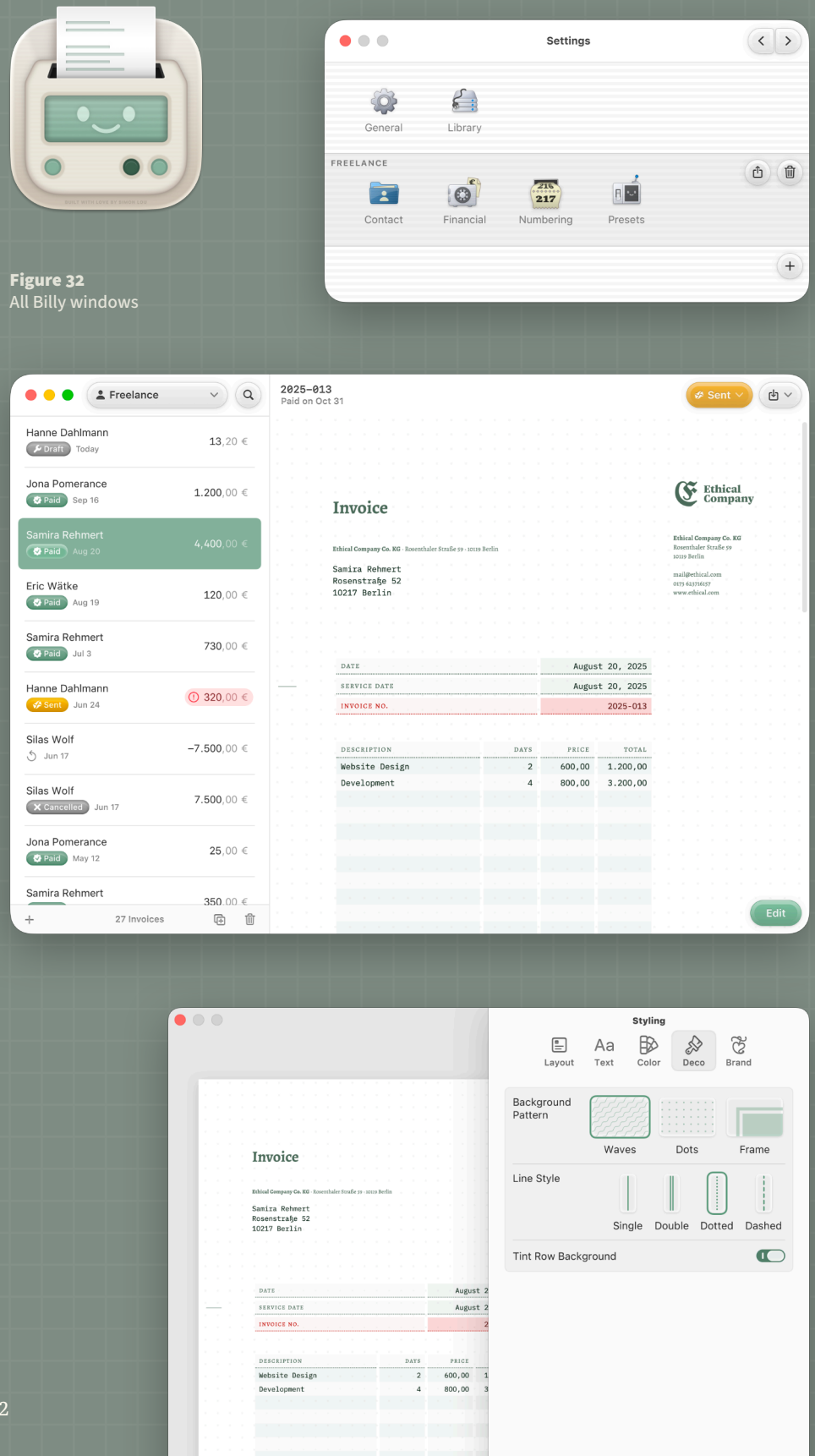



**Figure 32**
All Billy windows





CHAPTER 6

# *Designing with an Opinion*

As software always lives in context, I will demonstrate my design process using a real-world example—the invoice management app 'Billy'. Billy's purpose is to let freelancers and small business owners create, manage, and archive their invoices without having to master a complex new tool—simplicity is at its core. Additionally, it offers a limited set of tools to design and brand invoices that align seamlessly with the corporate identity of the user's company, while still ensuring readability and clarity.

I will take you on the journey of designing an app for the Mac in 2025, that is easy to use while providing features and flexibility for power users. My focus is on creating a Mac-assed app that is not shy of having its own identity—its own voice. I will touch upon questions, struggles, and approaches I came across while developing this example. This will not show a full chronological design process, but rather fragmentary insights.

## 6.1 Justifying Design Decisions

*Rationalizing intuition*

Designing an app means constantly facing situations where different requirements pull in different directions. Many decisions involve balancing technical con-

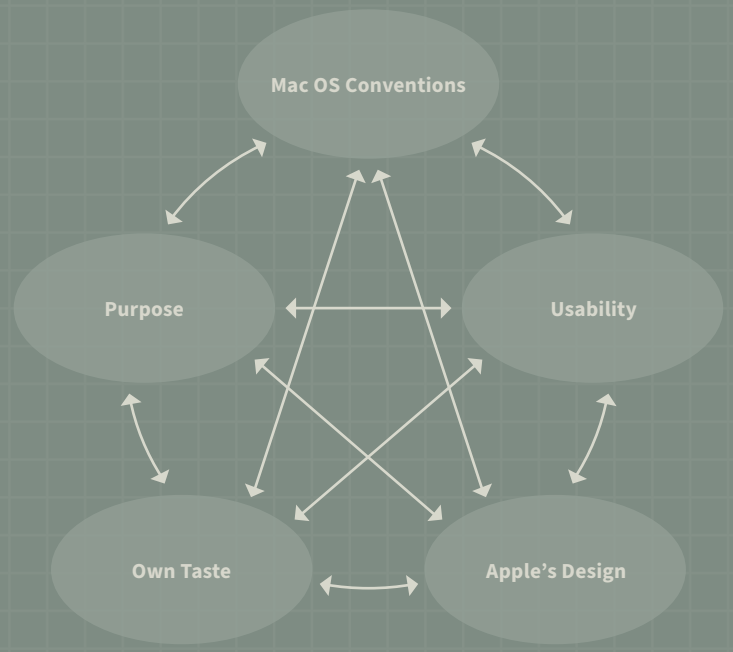



**Figure 33**
Design decision framework





straints, user expectations, aesthetic considerations, and the context of Mac OS, in which the app will be used. To navigate these challenges, I identified a set of aspects that I considered most relevant and used them to guide how I approached each design choice.

The first aspect was the app's purpose. Every decision had to support what the app is actually meant to do. Focusing on purpose helped me avoid solutions that looked appealing on the surface but did not meaningfully contribute to the app's function.

Another important factor was Mac OS' heritage and conventions. Mac OS comes with a strong set of expectations regarding how windows behave, how navigation works, and how everything should be integrated with the system. I considered these conventions because they shape what users are prepared for and what feels natural in this environment.

General usability also played a central role. Many design decisions came down to making controls clearly understandable, text legible, or evaluating how much friction is appropriate. Balancing usability with other goals—especially visual or structural preferences—was often one of the core challenges.

Of course, I also had to consider the current Apple design style. Apple's ongoing design evolution influences what feels 'modern' or appropriate on Mac OS. By acknowledging these trends, I tried to evaluate how much you can diverge from it without the app feeling disconnected from today's platform landscape.

Finally, I included personal expression and taste as a factor. Even within all the constraints and expectations, there is still room for individual interpretation. Allowing some space for my own design preferences helped the app develop a character and identity of its own, without overriding the other considerations.

These aspects reflect the elements I found most important when navigating the various challenges involved in

designing an app. They are not a general set of factors to consider for every app—just the ones I chose to balance to reach thoughtful and well-reasoned design decisions.

## 6.2 Challenges While Adopting Liquid Glass

*Designing for macOS Tahoe*

Most of my observations have focused on strategies from the past, which is understandable since those are the ones we've already observed and can learn from. Nevertheless, the recent developments in the Apple UI world have to be taken into consideration. The structural and visual changes in macOS Tahoe are higher than any since, arguably, iOS 7 (CUNNINGHAM, 2025). The question to discuss now is how to respond to the changes. Have the variables for creating a unique Mac experience changed? In the following, I will explore various strategies to deal with this new visual language called Liquid Glass.

App makers find themselves in a peculiar predicament, forced to choose between fully embracing Apple's design direction or crafting an interface that is inherently understandable through a logical structural hierarchy. What further complicates the decision is that the new default controls provided are in a state of quality that does not live up to the standard a lot of developers set out for themselves. Having to decide between a well-crafted UI and following Apple's vision is a new phenomenon (GRUBER, 2025).

According to Louie Mantia, a long time member of the Mac community, an app has to "match [the] level of visual quality [of the system]" (GRUBER, 2025). Even if an app does not fully adapt to the Liquid Glass system's appearance, it must still speak the same language. With such a distinctive language, that can be quite challenging and restrictive.

DESIGNING WITH AN OPINION

### 6.2.1 Strategies Regarding Visual Design

In relation to originality and expression, the approach for building apps for macOS Tahoe has changed in comparison to how app makers addressed these issues before. Just focusing on the design of custom controls got increasingly difficult. As a matter of fact, a lot of developers removed custom controls and went back to the system defaults for fear of feeling not native anymore (ARMENT & SMITH, 2025). Some of the motion behaviors are so intricate in the default controls that they are really difficult to replicate in a custom control.

#### Designing without Liquid Glass

An alternative is not adopting Liquid Glass and the layout changes it brings. That raises both obvious and less obvious issues. Once a design language has been established for the platform, users expect this visual style. Any deviation can quickly lead to disappointment or uncertainty.

On a more technical level, there are other considerations. Much of what people expect in terms of how a control behaves is already integrated into the standard controls. If you overwrite these interactions with your own, it is important to consider the behavior that people are accustomed to, to reproduce it, or at least to only consciously break with it. What needs to be considered now is that all custom controls will, of course, not receive any updates in the future. This means that the app developers themselves are responsible for ensuring that controls work on new operating systems and that the visual style continues to fit into the environment.

#### Designing with Liquid Glass

Apart from the issues with Liquid Glass discussed throughout this thesis, it is still a very impressive attempt at implementing the real-life visual behavior of

materials as complex as glass or water. Although its current use still has many problems, it can still be intriguing to integrate this material into your own design process. As Apple's implementation shows, it can be quite challenging to do it in a way that does not compromise important aspects like legibility or the sense of hierarchy. Maybe without the goal of showing off the 'shiny new thing', it is possible to use it in a way that it gives the design a nice touch—treating Liquid Glass as one tool in a designer's tool belt and not as the whole story. On the same note—other skeuomorphic materials could have the same relevance too, leaving Liquid Glass as one of many to choose from for giving an interface the right tone for the topic. Exactly this is the way I went with my app Billy—incorporating Liquid Glass at places where it make sense.

Maybe Liquid Glass does not have to be exclusive to the purposes of controls and Sheets and can be used differently on a per-app basis, as each app brings really different demands.

When designing custom controls with or without Liquid Glass, a lot of issues emerge that I discussed in the previous strategy.

### Designing on top of Liquid Glass

Liquid Glass acts as a weird mixture between a base layer where controls sit on top of and as the control itself. The elements that sit on top of it, like typography, iconography, or other UI elements, can be customized as well. Introducing a custom icon set or even a custom typeface can give an app a lot of individual identity without even touching the Liquid Glassy base layer below.

### Adapting the Style of Liquid Glass

One last imaginable approach is to embrace the style that Liquid Glass brings with it and refine it yourself. This can happen out of the motivation that the current state does not meet the quality standards that the developers have set for themselves. A prime example of this approach is Sketch (see *5.1 Sketch*), which introduced a toolbar entirely in the style specified by Apple, but did not use a default toolbar and instead built its own controls from scratch, including the implementation of the glass effect—a little less intrusive and ensuring legibility.

This approach requires a great deal of sensitivity. If the custom implementation behaves even slightly strangely or differently, the entire app experience will feel slightly off—not belonging to the Mac.

### Focussing the Visual Expression on the Content

Apple clearly states that their introduction of Liquid Glass is "making customizations [of controls] unnecessary" (HRISTOFOROVA, 2025, 8:33). Their perception is that the identity of an app can and should solely come from its content and not its controls.

That idea only fits a small category of apps—an example could be a news app, where the content and its presentation naturally define the feel of the product. But on the Mac, that is the exception, not the rule. Most Mac apps deal with constantly-changing user data, documents, graphics, or images, so the content cannot express identity because it is the variable, not the constant. What people actually recognize and rely on is the stable layer of controls and tools. In the majority of software, the interface is not a distraction from the content—it is the core of how the app works and the only consistent element that can carry its identity.

## 6.2.2 Building Toolbars

According to Apple a few years ago, "a toolbar gives users convenient access to the most frequently used commands and features in an app" (APPLE, 2013A, P. 178). This implies that a toolbar should be excluded from

hierarchical navigation, as it should always be visible because it contains controls that impact the entire application—at least that is how it used to be.

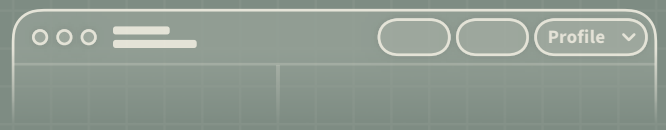
With macOS Tahoe the use and behavior of toolbars change. Controls sit now on glassy looking carrier shapes that house either a singular or a small group of controls.

Especially for more complex apps like Final Cut X, Logic, Sketch, but also something like Pages, the changes Liquid Glass brings result in two issues:

For more complex apps like Final Cut X, Logic, Sketch, but also something like Pages, the changes Liquid Glass brings result in two issues:

1. The controls become more spacious, allowing for fewer of them to occupy the same area within the toolbar. As a result, complex apps in particular cannot manage with the same amount of space as before and need to rethink the structure of their controls. For instance, a solution is the contextualization that the app Sketch chose. Depending on the current state, the toolbar items vary. In the case of Sketch, these variations primarily depend on the currently selected object. When no object is selected, a different set of actions appears compared to when a graphic is selected, and yet another displays when an image is selected. This ensures effective use of space and is very well executed in this case, but it also reduces discoverability. If you haven't selected an object, you do not know what actions are available.
2. With free-floating controls that are not housed in a toolbar that is a single visual entity, it seems reasonable to expect that they affect the underlying content directly. The perceived scope changes. Without the solid fill of the toolbar, the controls appear to be floating above the content, signifying a relationship between the control and the underlying content. This is based on the 'proximity principle', which

Traditional toolbar




Segmented toolbar

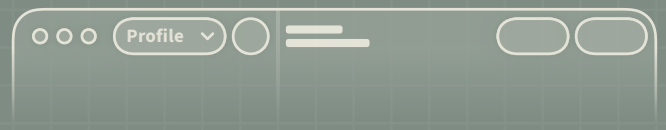



**Figure 34**
Billy toolbar options





states that elements near each other are perceived as related (HARLEY, 2020). As a result, the control is perceived as local rather than global. In some instances, that does not matter because the controls actually only influence the underlying content, as in Sketch. However, in other cases, it can be quite misleading. There are even situations where it cannot be foreseen what content will be below, as when the main app window is constructed out of columns—which control overlays which column changes with the window scaling. When the window is divided vertically, the problem becomes even more severe because toolbar items will never float over the bottom areas, yet they could potentially affect those very areas. In essence, having to ensure that toolbar controls overlay the content they influence significantly alters the design approach of toolbars and increases the likelihood of errors that would lead to user confusion.

The Billy app is a good example to illustrate the second issue using a real-world scenario. In a draft for macOS Sequoia, the toolbar included a profile switcher, which affected the whole app because all displayed invoices are part of the currently selected profile. Having this switcher floating on the content below the toolbar, in this case the currently selected invoice, suggests that it will only affect this invoice. To address this issue, there are only two options. The first is to split the toolbar into segments that correspond to the columns of the UI below, ensuring that each toolbar control is clearly associated with one content area. However, this approach raises questions about the toolbar's customization, as it would allow freely movable controls, potentially leading to misleading positions. The second option is to revert to a traditional separated toolbar, which does not have these issues but does not align well with the visual language of macOS Tahoe and the newly introduced toolbar conventions.

### 6.2.3 Lessons to Learn

When designing Mac apps with the assumptions and values described in this thesis, the introduction of Liquid Glass poses two difficulties:

1. How is it possible to stay true to a typical Mac behavior without feeling less integrated with the system than apps that fully adopt Liquid Glass layout conventions?
2. How can controls and surfaces get their own identity without just replacing the Liquid Glass material?

While there are aspects of typical Mac behavior that were changed dramatically, like toolbars, there are others that did not change at all. Even if Apple is currently not focusing on building really good experiences for the Mac, which materializes in their increasing use of Mac Catalyst for their own apps (COLUCCI, 2023), it is still possible to do that as an individual app maker. Things like more complex window layouts that are not just columns, the use of multiple windows in an app, or the other deeply rooted behaviors can still be utilized. Whether it is the right decision to adopt the new toolbar conventions is really dependent on the specific use case and, with it, the extent to which it would affect clarity and ultimately the usability of the app.

In a way, the approach of crafting original apps for the Mac totally changed with the release of macOS Tahoe. Before it was common to give your controls a custom look—the Mac identity came through structure and navigation. Now the visual identity of the system comes through liquid animations and the very unique new Liquid Glass effect on the controls themselves. When moving away from this glassy optic and the fragmented control placement, it quickly feels off—an alien in the system.

What will always stay is the option to incorporate original ideas, one-off navigation solutions, and humorous details to give an app character.

## 6.3 Building truly for the Mac

*Creating an app that understands and honors the platform*

Building an app that is meant for the Mac platform means balancing a solution that fits its purpose and Mac conventions. It is important not to reinvent things for which there are well-established solutions on the Mac. For example, settings are traditionally saved automatically—unless there is a compelling reason not to, it makes sense to adopt this approach.

In addition, many typical Mac behaviors do not conflict with other design decisions but are additional features that can be implemented—things like opening an item in a new window with a double-click.

### 6.3.1 Information Architecture

Generally, the app is structured around a simple information hierarchy. There must be a collection of invoices that can be individually previewed. To accomplish that, I decided in favor of a drill-down navigation with a list of invoices on the left and a big preview area on the right. That way, the user can browse the invoices while looking at the currently selected one.

Additionally, the collection of invoices is part of a profile, of which there can be multiple. Each profile has its own collection of invoices. The typical way to provide this higher level in the information hierarchy would be to add a source list—meaning an additional sidebar at the left. For example, the Apple Mail app went with this solution for mailboxes, and it is a very clear navigational method. I decided against this because a source list would occupy a lot of room, while most of it would stay unused because most people will either have only one or two profiles. The path I chose is to incorporate a profile picker in the form of a drop-down button at the top of the invoice list, which clearly shows its relation to

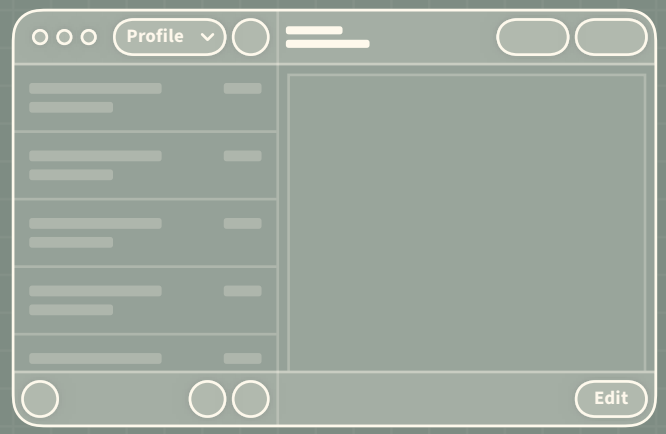



**Figure 35**
Information hierarchy of the Billy app

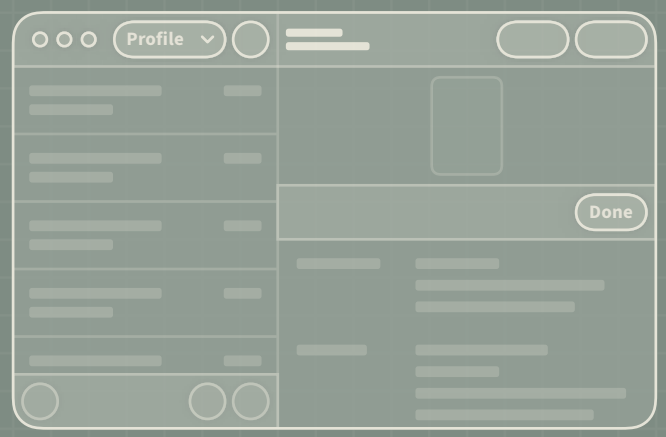



**Figure 36**
Edit drawer within the Billy app

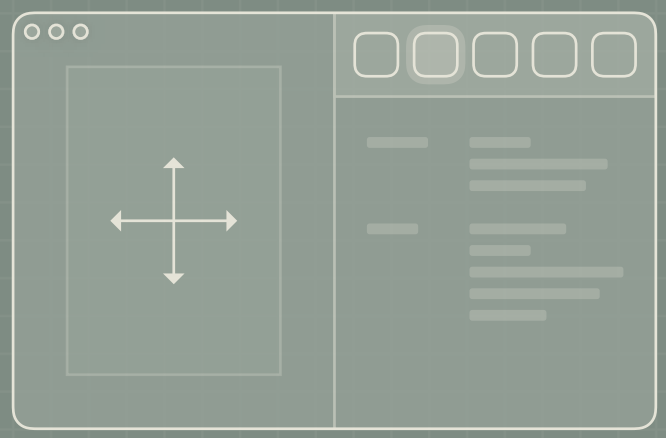


**Figure 37**
Invoice styling within the Billy app





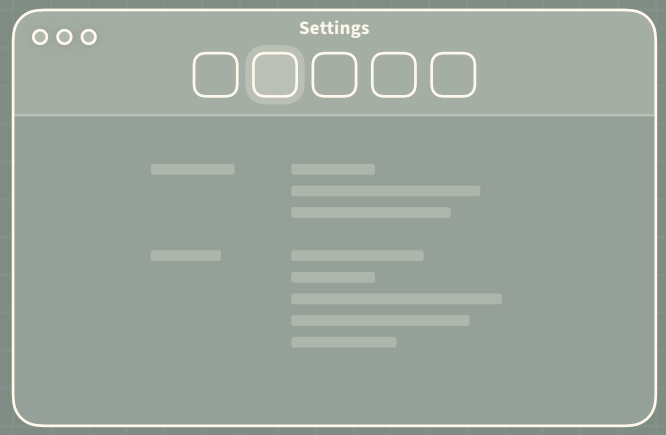
the invoices, gives quick access to it, and only occupies a fraction of the space. If I decide to add additional features like 'time tracking', 'expense cataloging', or 'offer creation' to the app in the future, I might reconsider this solution. These features would be a good fit for a source list. However, for the current range of functions, this solution works well.

Apart from previewing invoices, the primary objective of the application is to edit. Since most of the time, users select invoices to view rather than edit them, I decided to set the default state as a static preview. The edit form is accessible through a button in the bottom right. The initial idea was that the button would sit in a bottom bar, which was once a very typical UI element on the Mac. With clicking, the bar would raise, pushing the document to a minimized state and revealing the edit form. This behavior shows that the edits you make there will directly affect the document you selected before, while not creating unnecessary friction. This form of sliding motion was inspired by the drawer navigational element, which was a distinctive UI element of many early Mac apps until OS X Tiger (APPLE, 2005, PP. 176–178).

Because visually separated bars in macOS Tahoe are less common, I explored the option to utilize a floating button to open the edit form. This gives that button further emphasis, which is, being the primary action, not an issue here. Decisions like these are a balancing act of not losing identity but still adapting to the surroundings to provide a pleasant experience. The bar would create a clearer visual structure, while a floating button would give it a more contemporary feel.

For the purpose of styling an invoice, I opted for a separate window. As the user would do this effectively only once—creating a sort of preset, which gets used for all invoices in one profile—this extra level of friction feels appropriate. Layout-wise, this function had specific requirements. While changing the style, you want to see the adjustments as they are happening. This resulted in a split view with controls to manipulate the style on one

Default




System Preferences

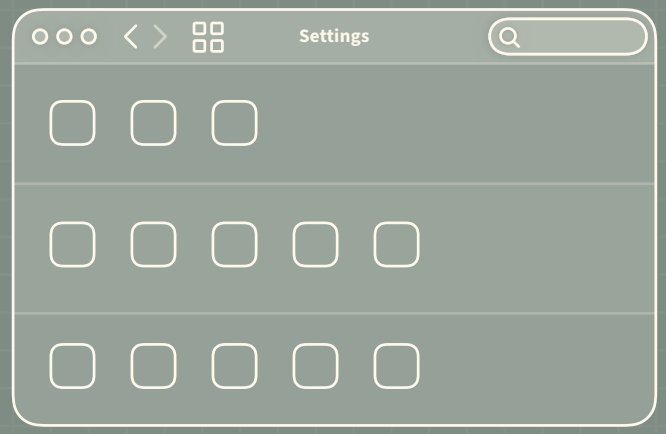



**Figure 38**
Settings layout options





side and a live preview on the other side. With the realization that most controls only affect one aspect of the invoice, I opted for a variable position and zoom state of the preview to best show the current adjustment. On the control side, I introduced a tabbed view—similar to how settings are structured traditionally—to cluster different styling options into categories.

## 6.3.2 Settings and Presets

As you would expect on the Mac, the settings open via a menu command and open in a separate window. The most common approach would be to segment the different settings into separate tabs. However, for this specific app, I opted for a different layout inspired by the System Preferences app in earlier Mac OS versions. This layout features a grid of large icons, each opening a subpage with a set of settings. This structure is generally suitable when dealing with a large number of settings that wouldn't fit in a single row of tabs. In my case, since I have a relatively small number of settings, I took advantage of another possibility this layout offers. I have general cross-app settings and multiple profiles with individual settings. Therefore, I use each row in the icon grid to represent a new profile. This allows me to have an overview of the general setting options and all profiles at the same time when opening the settings.

## 6.3.3 Data Management

Deciding on an approach to data management comes down equally to the specific use case of the app and to the philosophy of the developer. While different approaches make sense for different situations, often there remain many possible solutions that prioritize different aspects.

It was quickly clear that all invoices had to get stored in a single location. A completely document-based setup, where you would open an invoice, edit it, export it, and close the document, wouldn't work because all invoices had to exist in context with each other. Invoices had to

be sequentially numbered, which meant that the number of a new invoice depended on the last one. The logical solution was to have all invoices stored in a library accessible with just opening the app. All modifications of invoices would usually take place within the application.

Based on the simplicity of using it in the context of Apple's development environment and the comprehensive integration within the Apple ecosystem, I decided to use the SwiftData framework, built by Apple to store data persistently within an app. Technically, this is a database, which stores all data in one big file, hidden deeply in the library of the computer. A first prototype built using that approach worked technically fine—data is stored reliably and performantly.

The drawbacks of this decision I only later realized. There was no option to access the data in a way that is not intended by the app. The data—which belongs to the user—is owned by the app, which means users cannot freely use it in any way. There would be no way to use the data in a way that is not considered by the developer. Of course, that would be all solvable, for example, by providing excessive export and import options, but in my eyes, the whole approach is hostile design rather than user-friendly. I reconsidered and decided for a file-based solution where all invoices are stored as individual files. I chose the JSON file format as it is widespread and supported. The structure of the JSON I created myself as there are no existing conventions that fit this invoicing use case. This way, users can store their data where they want, by default in the library of the Mac, tucked away not to confuse less tech-savvy people, but easily movable via the settings of the app. Users can decide where to store and archive their financial data and even create and manipulate invoices directly on a file level of the Mac, not only inside Billy.

It can lead to a lot of frustration when an app stores data in a way that it cannot be backed up, transferred to a new device, or used with other software. Especially the last aspect is more based on my personal values, but nonetheless in my design process very important.

As the Mac always was the most powerful device in Apple's lineup, it always had a slightly different relation to control and customizability—I even see that as part of its identity and as a factor that sets it apart from other devices (see *3.3.1 Affordances of the System*). Although Apple has always had a foible for closed systems and limited freedom, the Mac allows deep intervention in the system—very different from the iPhone or iPad. Part of that is accessing the library of the device, which lets you see, manipulate, and control the data an app stores. What that communicates is a fundamental and very important value in software design—control over your own data. Building native software provides the chance to support that with the possibility to access the local file system and store created files locally. As these considerations are very close the technical implementation, they are not typically part of a design process. I think they should be.

## 6.3.4 The Right Amount of Friction

A lot of modern software aims to reduce friction as much as possible—making it faster to fulfill a certain task or spend money more easily in an online shop. In many cases, it makes sense to minimize friction, but in some, the friction has a purpose. Friction in the right moments can ensure that you do not accidentally delete a file, send an email, or perform any other irreversible action. Especially when dealing with tasks regarding the saving of documents, manipulating data, or deleting files, I intensely evaluated what the right amount would be. For instance, an interesting question was the behavior of the edit form. In a not really document-based environment, would people expect that this form would continuously auto-save, or just save as soon as the sheet gets closed? Is the option 'discard changes' needed? The more Mac-like behavior would certainly be an 'edited' state where the invoice is not saved automatically. However, at a time where people are accustomed

to tools such as Google Docs, many people certainly expect automatic saving behavior. The question comes down to whether the user sees the invoices as individual documents or as items in a library.

Although invoices are stored individually (see *6.3.3 Data Management*), I decided against treating them as individual documents in the UI. I anticipated that an autosave solution would prevent more errors than a solution with more user control would offer advantages. In this case, the technical decision does not have to influence the story the UI tells when there is a way that feels more logical to the user.

# 6.4 Coloring Outside the Lines

*How to design for a system so tightly regulated?*

As discussed at length over the course of this thesis, it is important and, at the same time, very hard to find and establish one's own style in a Mac app without compromising other important aspects like the native feeling of an app or its overall usability. There are a lot of decisions to make and and very 'gray areas' that demand good taste and a feeling for the platform.

It is not just a question of whether to adopt Apple's current design language or not, but also of building on it. Billy tries using Liquid Glass as a material and still design my own buttons and other controls that are clearly derived from Apple's but at the same time distinct. The goal is to create a unique identity while remaining compatible with the current environment—and thus Apple's design language.

## 6.4.1 Resisting Homogenization

With the observation that software is becoming less fun, diverse and simply boring, I tried resisting that notion and finding ways to incorporate delightful details and a unique voice that is allowed to feel different. I tried reinterpret existing patterns in a way that it does not feel confusing to the user while bringing something new to the table. My controls follow the effort doing exactly that—taking the existing shape and interpreting it differently while staying true to the established paradigm.

## 6.4.2 Finding Room for Expression

Despite the comprehensive design system and Human Interface Guidelines provided by Apple, there are many details that are not defined—after all, each app's demands are in some way unique. Dealing with very specific problems is a cornerstone of software design and always inherits the potential of creating unique, elegant, and custom solutions to these problems—a way of bringing originality to an app without breaking with any conventions.

While there are uncountable other ways to make an app stand out, it is important to find focus and not overdo it. Otherwise, the efforts to make an app feel at home on the platform fall short. For this specific project, I focussed on creating custom controls, designing colorful icons, and rethinking navigational interactions and behaviors while respecting Mac conventions.

## 6.4.3 Finding Inspiration

If you were a Mac user at that time, you probably still find some joy looking at Mac software from around 2010—many people argue the Mac was in its best state at that time. Surely a part of that is nostalgia, but I dare say that there are other reasons too.

To fully comprehend and learn from the design practice of the past, I rebuilt my app based on OS X Leopard. I believe Leopard is modern enough to address our current needs while still retaining the characteristics and remnants of earlier versions, such as OS X Tiger or Panther.

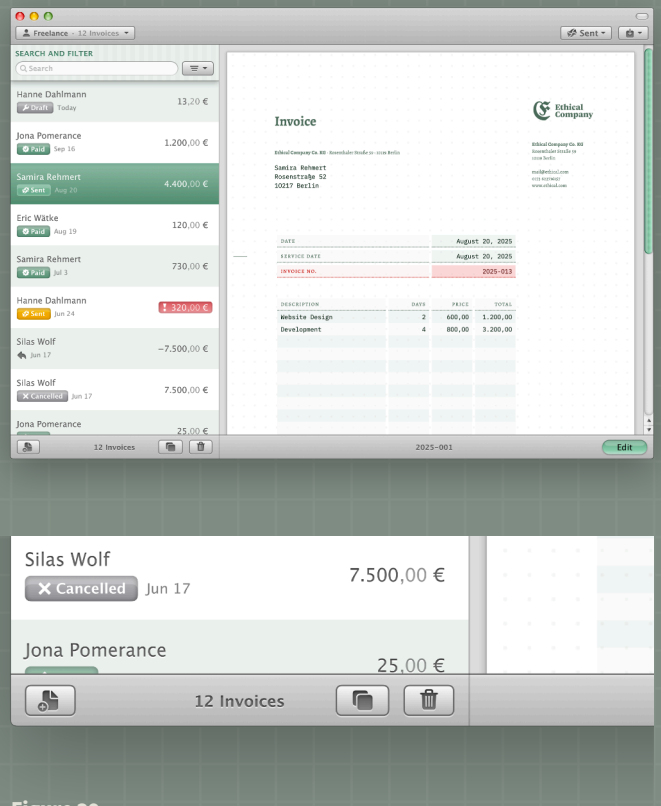



**Figure 39**
Billy inspired by Mac OS X Leopard





Amongst many well-thought-out details you only would notice when trying to recreate something, I rediscovered the level of care typography and icons received. Not only are icons aligned with the pixel grid to look perfectly crisp, but all typography got a treatment consisting of an inner and an outer shadow to create the illusion that the words are embossed into the background. This effect emphasizes what is a control and what is not—it creates a sense of permanence. Details like this add control and variation to the toolbox of a UI designer and are not just decorative but add clarity to the design.

## 6.4.4 Designing with an Opinion

Finding your own voice and the right style for a scenario can be challenging. For me, exploring how apps are currently approaching these things (see *5 Looking at Examples*) and then looking at the past and software from the 2000s and 2010s showed me how diverse software can be, but also what we have lost.

Finding one's own, contemporary style can be quite challenging as there are so many constraints in the form of conventions, guidelines, and design systems. The following will elaborate on the solutions I found in the areas that I previously defined.

### Navigation and Behavior

Manipulating the behavior of navigational elements can be tricky as it quickly leads to an unfamiliar feeling for the user. I made the interaction of opening the edit form a custom solution, as I couldn't find a common Mac pattern that would provide a sufficiently frictionless interaction that is at the same time clearly attached to the previewed invoice. In this case, a custom behavior will not throw users off because a button in a bottom bar is not a standard Mac convention and therefore not associated with a conflicting behavior. I am using the situation where I am unsatisfied with conventional


**Figure 40**
Button decision matrix





solutions to incorporate my own touch—in this case it grew to a signature element of the app.

## Controls

Traditionally, controls are the most common thing to customize in a Mac app because they are clearly defined elements that can change style while still being clearly recognizable for what they are.

Earlier in this chapter, I wrote about the level of detail of the system and that an app's design has to match the system's level of visual complexity to not feel out of place (see *6.2 Challenges While Adopting Liquid Glass*). To match the system and hit the level I want to achieve, I created an evaluation matrix. From left to right, the level of visual sophistication rises—on the left being very flat and close to the system's default for a push button, and on the right more illustrative and closer to the real-life object that is referenced—a keyboard key. Vertically, the roundness changes, with the bottom row being the maximum, like toolbar controls in macOS Tahoe would be. The middle button in the bottom row strikes a balance between maintaining its haptic feel and being clearly identifiable as a button while still functioning in the current Mac environment.

Especially in the middle area, I incorporated the Liquid Glass material, which gives it a slightly glassy behavior in situations where it floats over content and stylistically ties it in with the system.

While having a different execution, I chose an approach similar to Sketch's to deal with Liquid Glass (see *5.1 Sketch*)—using the new material and layout conventions, but incorporating it in a custom implementation.

## Icons

One aspect that falls short in modern Mac OS design is the use of unique iconography. *SF symbols* are a great basis and a well-made icon set, but it is not possible to

SF Symbols are Apple's official icon system, used throughout all Apple platforms and are also available to third-party developers.

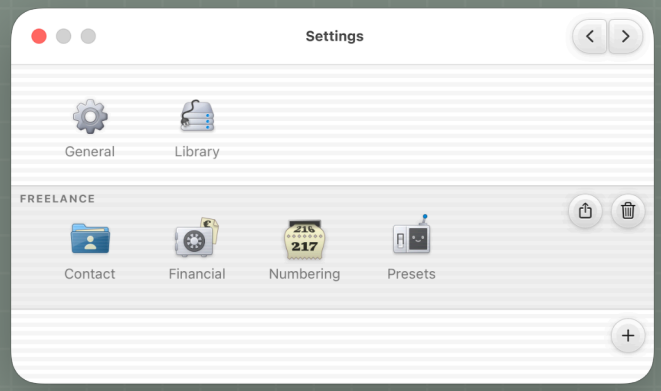



**Figure 41**
Billy settings window

find a perfectly fitting icon for every situation, nor does the style always fit. There are situations where a custom icon with its own personality can work better.

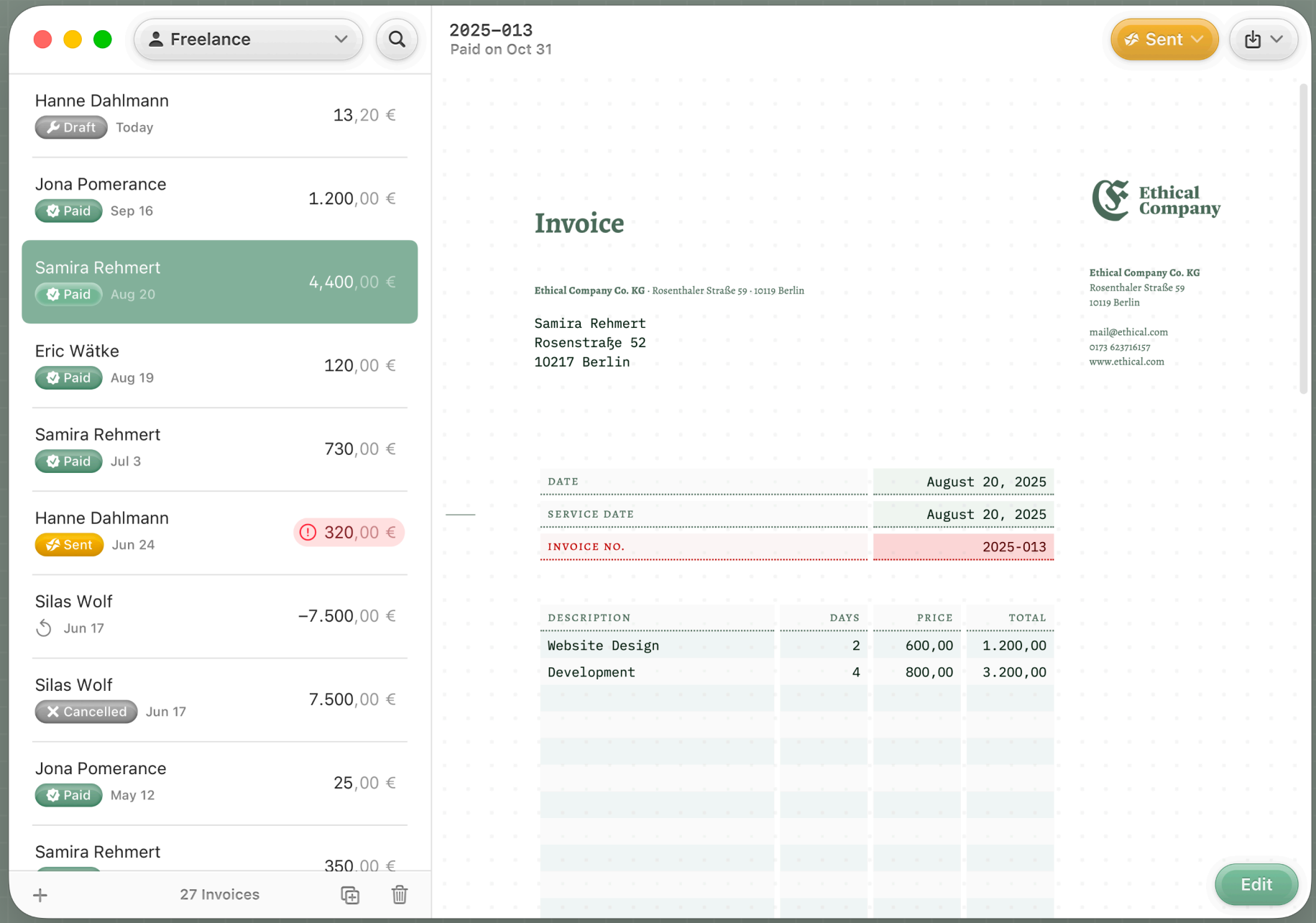
A noticeable positive example is Sketch (see *5.1 Sketch*), especially in their versions before macOS Tahoe was released. Crafting your own icons that fit in well with the system but still communicate exactly what you need is a great way to add quality and influence over the style without feeling out of place in the environment.

Nova (see *5.2 Nova*) took a very different route. Although system-provided options were available, they decided to integrate custom icons that are also very distinctive and expressive. They use colored icons with a very unique design style. They become a recognizable feature of the app and shape its identity.

As my app did not require very specific icons, and I was able to find very fitting SF symbols, I went with them for the controls. The settings were different. There were no suitable SF symbols for some pages, and in combination with the grid layout, something more stylistically expressive was needed. In the legacy of the old System Preferences, I designed custom colored icons that better captured their meaning. As settings are something that is sometimes considered a little bit boring and only opened from time to time, it felt appropriate to go with a style that would feel a little bit over the top in the main app window which the user sees every day.

DESIGNING WITH AN OPINION

**Figure 42**
Billy window



DESIGNING WITH AN OPINION

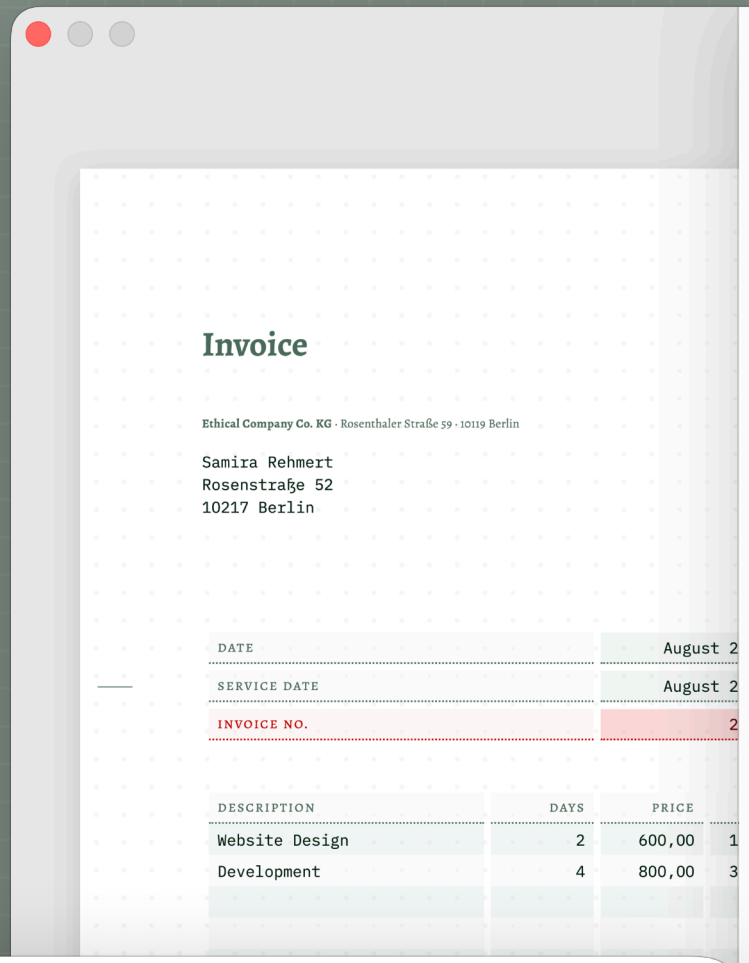



**Figure 41**
Billy settings window




**Figure 43**
Billy styling window

CHAPTER 7

# *Conclusions, Thoughts and Nostalgia*

After now four decades of visual software design, we have lived through multiple cycles of experimentation, correction, and consolidation. Each shift brought new possibilities, but also new excesses. What distinguishes the current moment is not merely another stylistic change, but a consolidation of visual authority. With Liquid Glass, Apple promotes a direction that prioritizes ecosystem unity over individual expression, implicitly encouraging conformity across applications. In doing so, it risks flattening the diversity, clarity, and character that historically defined the Mac platform.

> *I kept asking myself 'why?' as I used iOS 26 and MacOS 26 this summer.* (HEER, 2025)

Asking 'why?' is no obstacle to progress; it should be the foundation of every decision. Design decisions do not always require extensive research, but they do require intention. In the case of Apple's recent design changes, particularly those affecting toolbars and visual hierarchy, the rationale often appears disconnected from user benefit. Several of these changes even introduce regressions in clarity, discoverability, and accessibility—areas Apple itself once defined as core principles.

Paradigms must change, and designers must adapt alongside them. However, adaptation does not mean

uncritical acceptance. The future of software design lies in questioning dominant directions, not merely implementing them. Blind compliance with guidelines risks contributing to a homogeneous landscape where individuality and usability are deprioritized in favor of visual consistency.

The Mac's identity is not rooted in a visual style, but in a set of affordances, behaviors, and conventions refined over decades. Understanding this foundation enables designers to evolve the platform without erasing what makes it usable and recognizable.

Despite these tensions, the Mac remains a platform that allows for creative freedom. As demonstrated through the explorations in this thesis, it is still possible to design software that feels contemporary while maintaining strong usability and a distinct voice. The challenge moving forward is not to resist change, but to engage with it critically—to ask why, to understand what is at stake, and to design with intention rather than obedience.

## On a Personal Note

*Big tech and indies*

We developed a world of unbelievably fast technological progress and arrived in the age of centralization through huge corporations. The digital space once characterized by freedom got bought piece by piece by companies like Google, Microsoft, or Apple—making money off people's data, off people's lives. This not only creates massive power conglomerates but also strips software of its soul—an app transformed into a product, detached from the tool it once was.

Although this all sounds terribly negative and pessimistic, the greed of large companies creates a unique position for small, independent app makers. We can build what large corporations cannot: software for a few people instead of software for everyone. Only this allows to build software with care, attention to detail, and a level of quality that would never withstand the pressure of serving a billion users.

We are constantly told that everything must grow, reach thousands of users, and change the world. Maybe that is wrong. Maybe we need small again—software built for a community rather than a corporation.

APPENDIX

# *References*


Allen, A. (2020, September 17). No More Boring Apps. *!Boring Software.* Retrieved November 13, 2025, from https://www.notboring.software/words/no-more-boring-apps

Allen, A. (2024). *Serious play* [Video]. Config 2024. https://www.youtube.com/watch?v=wBnIyD5I8mM

Apple. (n.d.-a). Apple reimagines the iPhone experience with iOS 14. *Apple Newsroom.* Retrieved December 15, 2025, from https://www.apple.com/newsroom/2020/06/apple-reimagines-the-iphone-experience-with-ios-14

Apple. (n.d.-b). *Human Interface Guidelines.* Apple Developer Documentation. Retrieved June 26, 2025, from https://developer.apple.com/design/human-interface-guidelines

Apple. (n.d.-c). *OS.* Apple. Retrieved December 15, 2025, from https://www.apple.com/os

Apple. (1983). *[Brochure Showing Apple Lisa].* Computer History Museum. https://www.computerhistory.org/collections/catalog/102634506

Apple. (1984). *[Apple Macintosh Advertisement].* Computer History Museum. https://www.computerhistory.org/collections/catalog/102640962

Apple. (2000). *Macworld Expo Keynote.* https://www.youtube.com/watch?v=eOAcAxiZ3qE

Apple. (2005). *Apple Human Interface Guidelines.*

Apple. (2013a). *OS X Human Interface Guidelines.*

Apple. (2013b). *What Is Cocoa?* Documentation Archive. Retrieved November 17, 2025, from https://developer.apple.com/library/archive/documentation/Cocoa/

Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html

Apple. (2020). *Keynote* [Video]. WWDC20. https://developer.apple.com/videos/play/wwdc2020/101

Apple. (2022). *Platforms State of the Union* [Video]. WWDC22. https://developer.apple.com/videos/play/wwdc2022/102

Apple. (2025). *Keynote* [Video]. WWDC25. https://developer.apple.com/videos/play/wwdc2025/101

Arment, M., & Smith, D. (2025, July 17). *The Tidal Wave* (No. 323). https://www.relay.fm/radar/323

Asghar, M. Z., Alam, K. A., & Javed, S. (2019). *Software Design Patterns Recommendation: A Systematic Literature Review.* 167–172. https://doi.org/10.1109/FIT47737.2019.00040

Atkinson, B. (2022a). *Bill Atkinson: Interview for Lisa's 40th Anniversary* [Computer History Museum]. https://youtu.be/NIJrCeRjZIU

Atkinson, B. (2022b). *Bill Atkinson: Polaroids showing the Evolution of the Lisa GUI* (H. Hsu, Interviewer) [Computer History Museum]. https://www.computerhistory.org/collections/catalog/102809056/

Atkinson, B., & Hertzfeld, A. (2010). *MacPaint Interview and Demonstration, with Bill Atkinson and Andy Hertzfeld* (A. Gardner, Interviewer) [Computer History Museum]. https://www.computerhistory.org/collections/catalog/102702313

Bentley, F., Feldman, J., Schmidt, L., Pielot, M., & Gilbert, M. (n.d.). *Better, Easier, Emotional UX.* Google Design. Retrieved September 23, 2025, from https://design.google/library/expressive-material-design-google-research

Bramhill, M. (2015a). *Itty-Bitty Buttons* (Extra). https://www.macintosh.fm/episodes/buttons

Bramhill, M. (2015b). *Rich Corinthian Leather* (No. 1). https://www.macintosh.fm/episodes/1

Chin, M. (2022, October 27). Apple macOS 13 Ventura review: a bunch of good updates you can mostly ignore. *The Verge.* Retrieved October 8, 2025, from https://www.theverge.com/23425407/apple-macos-13-ventura-review-mac-imac-macbook

Colucci, A. (2023, November 28). *Apple's use of AppKit, Catalyst, Swift and SwiftUI in macOS Sonoma.* Timac. Retrieved November 16, 2025, from https://blog.timac.org/2023/1128-state-of-appkit-catalyst-swift-swiftui-mac

Crampton Smith, G. (2007). What Is Interaction Design? In *Designing Interactions* (1st ed., pp. viii–xix). MIT Press.

Cunningham, A. (2016, September 20). macOS 10.12 Sierra: The Ars Technica Review. *Ars Technica.* Retrieved December 14, 2025, from https://arstechnica.com/gadgets/2016/09/macos-10-12-sierra-the-ars-technica-review

Cunningham, A. (2020, November 12). macOS 11.0 Big Sur: The Ars Technica Review. *Ars Technica.* https://arstechnica.com/gadgets/2020/11/macos-11-0-big-sur-the-ars-technica-review

Cunningham, A. (2025, September 15). macOS 26 Tahoe: The Ars Technica Review. *Ars Technica.* Retrieved November 14, 2025, from https://arstechnica.com/gadgets/2025/09/macos-26-tahoe-the-ars-technica-review

Delicious Monster. (n.d.). *[Screenshot of Delicious Library 3].* Delicious Monster. Retrieved July 25, 2024, from https://delicious-monster.com

Doosti, B., Dong, T., Deka, B., & Nichols, J. (2018). *A Computational Method for Evaluating UI Patterns* (arXiv:1807.04191). arXiv. https://doi.org/10.48550/arXiv.1807.04191

Evans, C. L. (2013, June 11). A Eulogy for Skeuomorphism. *VICE.* https://www.vice.com/en/article/a-eulogy-for-skeumorphism

Fessenden, T. (2021, April 11). Design Systems 101. *Nielsen Norman Group.* Retrieved September 23, 2025, from https://www.nngroup.com/articles/design-systems-101

Gates, B. (2025). *Source Code: My Beginnings* (1st ed.). Penguin Books Ltd.

Guzman, M. (n.d.). *Toolbar Guidelines.* Mario Guzman. Retrieved October 17, 2025, from https://marioaguzman.github.io/design/toolbarguidelines/marioaguzman.github.io/design/toolbarguidelines

Gruber, J. (2025, July 31). *"Michigan-Starred Fine Dining", With Louie Mantia* (No. 428). https://daringfireball.net/thetalkshow/2025/07/31/ep-428

Hackett, S. (2005). *[Screenshot of OS X 10.4 Tiger].* 512 Pixels. Retrieved December 8, 2025, from https://512pixels.net/projects/aqua-screenshot-library/mac-os-x-10-4-tiger/#jp-carousel-15702

Hackett, S. (2012). *[Screenshot of Contacts Application in OS X 10.8 Mountain Lion].* 512 Pixels. Retrieved December 8, 2025, from https://512pixels.net/projects/aqua-screenshot-library/os-x-10-8-mountain-lion/#jp-carousel-16116

Hackett, S. (2014). *[Screenshot of OS X 10.10 Yosemite].* 512 Pixels. Retrieved December 8, 2025, from https://512pixels.net/projects/aqua-screenshot-library/os-x-10-10-yosemite/#jp-carousel-16342

Hackett, S. (2020). *[Screenshot of macOS 11 Big Sur].* 512 Pixels. Retrieved December 8, 2025, from https://512pixels.net/projects/aqua-screenshot-library/macos-11-0-big-sur/#jp-carousel-20982

Hackett, S. (2025). *[Screenshot of macOS 26 Tahoe].* 512 Pixels. Retrieved December 8, 2025, from https://512pixels.net/projects/aqua-screenshot-library/macos-26-tahoe/#jp-carousel-33781

Harley, A. (2020, August 2). Proximity Principle in Visual Design. *Nielsen Norman Group.* Retrieved November 25, 2025, from https://www.nngroup.com/articles/gestalt-proximity

Heer, N. (2025, September 29). On Liquid Glass. *Pixel Envy.* Retrieved October 5, 2025, from https://pxlnv.com/blog/on-liquid-glass

Hertzfeld, A. (2004). *Revolution in The Valley: The Insanely Great Story of How the Mac Was Made* (1st ed.). O'Reilly Media.

Hodson, B. (2016, April 26). Rediscovering Apple's Human Interface Guidelines from 1987. *Prototypr.* Retrieved November 29, 2025, from https://blog.prototypr.io/rediscovering-apples-human-interface-guidelines-1987-59731376b39e

Hristoforova, M. (2025). *Get to know the new design system* [Video]. WWDC25. https://developer.apple.com/videos/play/wwdc2025/356

iA. (2018, December 14). iA Writer now offers three custom designed writing fonts. *iA News.* Retrieved December 10, 2025, from https://ia.net/topics/a-typographic-christmas

Isaacson, W. (2011). *Steve Jobs* (1st ed.). Simon & Schuster.

IxDF. (2016). *What is Visual Design?* The Interaction Design Foundation. Retrieved July 18, 2025, from https://www.interaction-design.org/literature/topics/visual-design

Kare, S. (1984). *Susan Kare demonstrating the Macintosh Interface in 1984* [Video]. https://youtu.be/ZmWOtf4Ziso

Kominski, R. (1991). *Computer Use in the United States: 1989* (Series P-23 No. 171; Current Population Reports). U.S. Census Bureau. https://www2.census.gov/library/publications/1991/demographics/p23-171.pdf

Kuts, E. (2009). *Playful User Interfaces: Literature Review and Model for Analysis.* Proceedings of DiGRA 2009 Conference. https://doi.org/10.26503/dl.v2009i1.492

Laws, D. (2018, September 20). Who Invented the Microprocessor? *Computer History Museum.* Retrieved October 26, 2025, from https://computerhistory.org/blog/who-invented-the-microprocessor

Loranger, H. (2015, March 8). Beyond Blue Links: Making Clickable Elements Recognizable. *Nielsen Norman Group.* Retrieved November 20, 2025, from https://www.nngroup.com/articles/clickable-elements/

MacDonald, T. (n.d.). *Get to Know the Tides Near You.* App Store. Retrieved December 13, 2025, from https://apps.apple.com/us/story/id1728306221

Moggridge, B. (2007). *Designing Interactions* (1st ed.). MIT Press.

Mrgan, N. (2013, May 10). [Button Style Test]. *Neven Mrgan's Blog.* Retrieved December 4, 2025, from https://mrgan.tumblr.com/post/50108095253/let-a-button-be-a-button

NeXT. (1995). *NeXTstep 3.3 Developer Documentation.* Retrieved October 28, 2025, from https://www.nextcomputers.org/files/manuals/nd/

Nicol, J. (2013, January 28). The demise of skeuomorphism. *Jonathan Nicol.* Retrieved October 26, 2025, from https://jonathannicol.com/blog/2013/01/28/the-demise-of-skeuomorphism

Norman, D. A. (2008). Signifiers, Not Affordances. *Interactions, 15*(6), 18–19. https://doi.org/10.1145/1409040.1409044

Panic. (n.d.). *Nova.* Retrieved September 23, 2025, from https://nova.app

Panic. (2025, May 22). *Building Extensions: Getting Started.* Nova Docs. Retrieved December 10, 2025, from https://docs.nova.app/extensions/getting-started

Perez-Cruz, Y. (2019). *Expressive Design Systems* (Vol. 31). A Book Apart.

Quinlan, A., & Hajas, P. (2025). *What's new in SwiftUI* [Video]. WWDC25. https://developer.apple.com/videos/play/wwdc2025/256

Rausch, F. (n.d.). *Modern iOS Navigation Patterns.* Frank Rausch. Retrieved July 4, 2025, from https://frankrausch.com/ios-navigation

Riddering, M. (Director). (2024). *What's wrong with software design today* (No. 44). https://www.dive.club/deep-dives/andy-allen

Scher, P. (2017). *Life Lessons from the Field.* beyond tellerrand, Berlin. https://beyondtellerrand.com/events/berlin-2017/speakers/paula-scher

Siracusa, J. (2011, July 20). Mac OS X 10.7 Lion: The Ars Technica Review. *Ars Technica.* https://arstechnica.com/gadgets/2011/07/mac-os-x-10-7

Siracusa, J. (2014, October 16). OS X 10.10 Yosemite: The Ars Technica Review. *Ars Technica.* Retrieved December 11, 2025, from https://arstechnica.com/gadgets/2014/10/os-x-10-10

Sketch. (n.d.). *Sketch Media Kit.* Retrieved December 10, 2025, from https://www.sketch.com/about-us/#press

Sketch. (2021a, October 14). Inside Sketch: how we redesigned our toolbar icons for Big Sur and Monterey. *Sketch.* Retrieved December 9, 2025, from https://www.sketch.com/blog/how-we-redesigned-our-toolbar-icons-for-big-sur-and-monterey

Sketch. (2021b, December 23). Tech Talk: How we made the Mac app faster with improved rendering. *Sketch.* Retrieved December 10, 2025, from https://www.sketch.com/blog/tech-talk-how-we-made-the-mac-app-faster-with-improved-rendering

Sketch. (2022, January 6). Open format: how to read Sketch files and convert to JSON. *Sketch.* Retrieved December 10, 2025, from https://www.sketch.com/blog/open-format-reading-sketch-file-sketch-to-json

Skowron, T. (2020, July 8). Printing with SwiftUI. *Thomas Skowron.* Retrieved December 3, 2025, from https://thomas.skowron.eu/blog/printing-with-swiftui

Snell, J. (2020, November 12). macOS Big Sur Review: Third age of Mac. *Six Colors.* Retrieved October 9, 2025, from https://sixcolors.com/post/2020/11/macos-big-sur-review-third-age-of-mac

Tidwell, J., Brewer, C., & Valencia, A. (2020). *Designing Interfaces: Patterns for Effective Interaction Design* (2rd ed.). O'Reilly.

Troughton-Smith, S. (2022, February 16). Where Mac Catalyst Falls Short. *High Caffeine Content.* Retrieved November 27, 2025, from https://www.highcaffeinecontent.com/blog/20220216-Where-Mac-Catalyst-Falls-Short

Voorhees, J. (2025, September 15). macOS 26 Tahoe: The MacStories Review. *MacStories.* https://www.macstories.net/stories/macos-26-tahoe-the-macstories-review

Walker, R. (2003, November 30). The Guts of a New Machine. *The New York Times Magazine.* Retrieved July 18, 2025, from https://www.nytimes.com/2003/11/30/magazine/the-guts-of-a-new-machine.html

Whitenton, K. (2013, December 22). Minimize Cognitive Load to Maximize Usability. *Nielsen Norman Group.* Retrieved October 28, 2025, from https://www.nngroup.com/articles/minimize-cognitive-load

Wichary, M. (1996). *[Screenshot of OpenStep 4.2].* Guidebook Gallery. Retrieved December 8, 2025, from https://guidebookgallery.org/screenshots/openstep42

Wichary, M. (2003). *[Screenshot of Mac OS X 10.3 Panther].* Guidebook Gallery. Retrieved December 8, 2025, from https://guidebookgallery.org/screenshots/macosx103

Worboys, C. (2024). *The broken promises of design systems* [Video]. Config 2024. https://www.youtube.com/watch?v=BQXTt-NZ2Bs

APPENDIX

# *Figures*

APPENDIX

# *Use of AI*

The following AI-based tools were used in the creation of this thesis:

- The Apple Intelligence ‘Proofread’ function was used for proofreading purposes only (grammar and spelling), without changing the content or meaning of the text.
- OpenAI’s GPT-5.2 were used for research suggestions and recommendations for literature.
  *Example prompt: Which literature or paper could I read on the topic […]?*

The content of all outputs from Apple Intelligence and GPT-5.2 was checked.

APPENDIX

# *Declaration of Authorship*

I hereby declare under oath that:

- I have written this thesis independently,
- I have used no sources or aids other than those indicated,
- neither this thesis nor parts of it have been submitted elsewhere as part of an examination,
- and all passages that are taken verbatim or in substance from other works are clearly identified as such and properly cited in accordance with the generally accepted rules of academic citation.

Potsdam, December 16, 2025

*Simon Lou Schüler*

*Thanks for the Inspiring Conversations to*
Boris Müller
Chris Nuzum
Frank Rausch
James Thomson (Email)
Jona Pomerance
Louie Mantia (Email)
Mario Guzman
Samira Rehmert
Thibault Guérin

*Proofread by*
Chris Nuzum
Jona Pomerance
Samira Rehmert

*Fonts*
Instrument Serif by Rodrigo Fuenzalida & Jordan Egstad
Nice by Jan Fromm
Source Sans by Paul D. Hunt

*Printed by*
Buch- und Offsetdruckerei H. Heenemann

*Good design doesn't have*
*to be beautiful—it has to be personal.*

JESPER KOUTHOOFD, 2024