The Design of Everyday Shiny Apps

Casey Aguilar-Gervase, Atorus Research, Indianapolis, USA
Maya Gans, Atorus Research, Las Vegas, USA

ABSTRACT

Donald Norman's *Design of Everyday Things* reminds us that even intelligent, capable people struggle with poorly designed objects—pushing a door that should be pulled or flipping the wrong switch despite clear intentions. The problem is rarely the user; it is the design that fails to communicate what actions are possible or what just happened. The same issues exist in Shiny apps, where functionality often outweighs experience. As Shiny developers, we are effectively the door designers, responsible for shaping how people move through data and decisions. The encouraging part is that good design is not reserved for formally trained designers. You do not need an art degree to build intuitive, visually coherent applications—you need practical principles and repeatable patterns.

Shiny apps often begin as proof-of-concept tools built to demonstrate that an analysis can run. When those same applications reach real users, however, teams discover a gap between *functioning* and *usable*. This paper describes a practical, design-first approach for closing that gap. Drawing from our experience as consulting Shiny developers, we show how four lenses—layout, color, typography, and components—can transform an existing application without changing its underlying functionality. Through a case study of a sample client app, we illustrate how small, intentional design decisions reduce confusion, build trust, and move a tool from "it runs" to "it's clear and ready for production."

INTRODUCTION

As Shiny developers in consulting, our job is to connect function with feel so client applications can move from "it runs" to "it's usable and clear." Most of the projects we inherit are not blank canvases—they are proof-of-concept applications built by researchers or data scientists who quite reasonably focused on getting the analysis right first. By the time those applications reach us, the request is almost always the same: *make this work better…and make it look better.*

Those two goals are really one goal. An application that looks confusing usually *is* confusing, and an interface that feels unreliable makes even correct outputs hard to trust. Rather than treating design as decoration applied at the end, we approach it as a set of practical decisions that shape how people understand and use the tool. In this paper, we describe the framework we use every day when turning POCs into production-ready Shiny apps.

We organize our work through four lenses. Layout determines whether a screen is scannable and predictable. Color directs attention without requiring extra words. Typography creates a hierarchy that can survive dense tables and forms. Components—tooltips, feedback messages, and controls—answer the user's constant questions: *What is happening? What changed? What do I do next?* These lenses allow us to make consistent choices even when the only design guidance from a client is a minimal brand kit.

THE INITIAL EXPERIENCE

When Shiny apps move from the developer to real users, a familiar set of issues tends to surface—especially when design direction from the client is minimal. Often the only guidance we receive is a short brand kit: five approved colors and a single typeface, with little explanation about hierarchy, interaction style, or tone. The expectation, however, is clear: transform a proof-of-concept into something that feels intentional and professional.

In that environment, the underlying functionality usually works, yet the experience feels uncertain. Users open an app and see useful information, but no obvious path forward. Color appears all at once because every “approved” color is used equally, so instead of guiding attention it competes for it. Navigation reflects how the code was organized rather than how a person thinks through a task, and sections do not always flow in an expected order.

Interaction design frequently reveals the same gaps. A user completes a form, and clicks *Create*, but nothing signals what changed. Buttons for updating, deleting, or downloading behave identically—silent—and the interface offers no hint about whether the system is loading, processing, or waiting for input. In these moments users are forced to guess: *Is it broken? Did I miss a step? Should I click again?* Even when the calculations are correct, the absence of feedback erodes confidence.

These problems are rarely about technical ability. They arise because most Shiny apps begin as analytical tools, not designed products, and because a short brand kit cannot on its own answer questions about layout, hierarchy, or interaction. Our role is to translate those few constraints—five colors and a typeface—into a coherent experience. We do that by reframing the task as a set of design questions: How do we make next steps visible? How do we reduce the need for memory? How can the interface speak back to the user? The four lenses introduced in this paper provide a practical way to answer those questions even when direction is sparse.

LAYOUT: MAKING STRUCTURE VISIBLE

When clients ask for an application to be “cleaner and more user friendly,” they are usually asking for better structure, even if they do not use that word. Our first step in any project is to ignore color and style entirely and look only at layout—how information is arranged and what path it suggests to a new user. Many Shiny apps inherit their structure from the way the code was written rather than from the way a person completes a task. Sidebars become catch-all containers, tables overflow their cards, and important context disappears as users move between pages.

Focusing on layout means asking simple questions. Can someone tell where to start without reading instructions? Does the screen have a clear reading order, or does the eye bounce between unrelated elements? Are related pieces of information visible at the same time, or does the user need to remember something from a previous step? These questions matter more than any specific component choice.

In practice, small structural decisions have outsized impact. Giving content the full width of the page can reduce the feeling of crowding. Consistent margins and an even grid help sections breathe and signal what belongs together. Nested navigation can separate exploration from review so that detailed tables do not overwhelm higher-level tasks. Keeping current selections visible alongside results prevents the constant loss of context that forces users to backtrack.

None of these change what the application can do. They simply make the existing functionality easier to understand. Layout becomes a quiet guide, turning a collection of features into a predictable flow that users can follow without thinking about the interface itself.

COLOR: GUIDING ATTENTION

Once a stable structure is in place, color can return as a tool rather than a distraction. In many Shiny apps, color enters by accident: defaults from packages mix with brand accents, and every available hue is used at equal volume. The result is visual noise where nothing feels more important than anything else. Color becomes decoration rather than guidance.

Working with limited client direction—often only a short palette and a typeface—requires translating those raw ingredients into a hierarchy. The first question is not *which colors do we have?* but *what jobs need color?* Navigation needs to signal location. Actions need to look clickable. Feedback needs to feel different from content. When color is assigned to roles instead of surfaces, even a small palette can create a coherent language.

Subtle repetition builds that language. Echoing an accent across navigation, cards, and interactive elements creates unity without adding clutter. Distinct hover states teach users what will respond to them. Replacing generic package defaults with brand-aligned tones makes tables and controls feel like part of the same system rather than borrowed parts. None of these choices alter what the application does, yet they shape how confidently a person moves through it. Used this way, color stops competing for attention and begins directing it. The interface no longer shouts; it points.

TYPOGRAPHY: ESTABLISHING HIERARCHY

Data-heavy applications live or die by typography. In many Shiny apps, text styles emerge from defaults rather than decisions. Headers, inputs, and actions blend together with little distinction. When everything carries the same visual weight, users must work harder to understand what matters.

Introducing a clear typographic system is often the most effective way to create order without adding new components. Even with a single client typeface, hierarchy can be built through deliberate use of weight, case, and size. Input labels can be treated consistently so forms feel structured rather than improvised. Navigation can carry stronger emphasis than surrounding content to signal where a user is. Small shifts in size can separate titles from metrics inside cards and tables, allowing dense information to be scanned instead of read line by line.

Typography also communicates intent. The way a button is styled can suggest whether it represents a primary action or a secondary option. Headers can anchor sections so users understand where one task ends and the next begins. These are subtle adjustments, but they reduce the cognitive effort required to parse each screen.

When type establishes a reliable rhythm, the interface begins to feel trustworthy. Users may not notice the hierarchy consciously, yet they benefit from it every time they can find what they need at a glance.

COMPONENTS: MAKING THE APP TALK BACK

The largest usability gaps in Shiny apps appear when the interface fails to communicate state. A calculation may be running, a file may be downloading, or a form may have been accepted, yet the screen remains silent. When feedback is missing, users are forced to rely on memory and guesswork, and even a correct system begins to feel unreliable.

Thoughtful components close that gap by turning one-way interfaces into conversations. Persistent hints or tooltips can provide a stable reference so users do not need to remember instructions from earlier screens. Contextual explanations at the section level can describe what an area is for and how it fits into the larger task. Controls that appear only when relevant guide people through a sequence instead of presenting every option at once.

Feedback is especially important around irreversible or time-consuming actions. Confirmation steps for destructive operations act as guardrails, while progress indicators and clear error messages reassure users that the system has heard them. Familiar, task-appropriate widgets—such as table components that behave like a spreadsheet—reduce the learning curve by borrowing patterns people already know.

These micro-interactions are small individually, but together they change the character of an application. The interface stops being a silent form and becomes a partner that answers the questions users naturally ask: *Did that work? What happens next? Am I safe to continue?*

CONCLUSION

It is easy to dismiss design changes as cosmetic, yet they shape whether an application is adopted and used correctly. When layout is confusing, people hesitate. When tone feels inconsistent, they question the results. When feedback is absent, they repeat actions or abandon tasks altogether. Design is therefore not about making something pretty; it is about making work possible.

The four lenses described in this paper offer a practical way to do that. Thoughtful composition—grids, whitespace, and hierarchy—helps users navigate without effort. Color and typography establish a tone that ties an application to its organizational identity and makes it feel familiar rather than improvised. Clear components and feedback reduce guesswork so people can act with confidence. None of these changes alter the underlying analysis, yet together they transform how that analysis is experienced.

Often it is the smallest details that matter most: a tooltip that clarifies intent, a disabled button that prevents mistakes, a spinner that acknowledges progress. These elements are inexpensive to implement, but they give an interface the quality Norman describes in well-designed objects—the sense that the system is understandable and on the user’s side.

Moving a Shiny app from proof-of-concept to production does not require new frameworks or abandoning existing code. It requires intentional choices about how the interface communicates. By applying layout, color, typography, and components as guiding principles, developers can create tools that not only run, but that users understand, trust, and want to return to.

REFERENCES

Norman DA. *The Design of Everyday Things*. Revised and expanded edition. New York: Basic Books; 2013.