ReportCaster: Redesigning a 35-Year-Old Scheduler (and Myself)

When I joined the WebFOCUS team, I didn't walk into a shiny new greenfield product. I walked into a forest.

Hidden in that forest was **ReportCaster** — a 30-plus-year-old scheduling tool quietly powering millions of reports for enterprise customers. It wasn't on the homepage. It wasn't in the sales deck. But if you talked to anyone who'd been around long enough, they'd say:

"Oh, if ReportCaster breaks, the business stops."

The UI looked like it had time-traveled from a Windows 95 screenshot. The workflows were scattered across different tools. The people who truly understood it lived in support, in engineering, and in long email threads — not in any product spec.

There was no PM deck, no Figma file, no documentation. Just a critical, legacy product… and a lot of pain.

Two weeks into my role, I raised my hand and said:

"I'll take this."

That was the beginning.

Walking Into the Dark: No PM, No Doc, Just a Critical Tool

On paper, this was not a safe project to volunteer for.

- There was no dedicated PM for ReportCaster when I started.
- No UX baseline, no previous designer to inherit from.
- Zero up-to-date documentation — most of what existed was stale or purely technical.
- The tool lived partly inside the WebFOCUS ecosystem and partly in its own aging UI, with workflows that jumped across multiple environments.

It really did feel like walking into a dense forest at night, with a flashlight whose battery might or might not be dying.

So I started with the only thing that made sense:

"If I can't read the product, I'll read the people."

I booked time with:

- Long-time engineers who had lived with this codebase for decades
- Support engineers who handled "fire drills" when something broke

- Internal power users who had built entire workflows on top of ReportCaster
- Product stakeholders who were hearing "we need a more modern scheduler" from customers but didn't know what that actually meant in UX terms

My goal wasn't to ask, *"What do you want ReportCaster to look like?"* My goal was to understand, *"Why do you still rely on this thing, despite how painful it is?"*

Meeting the People Behind the Schedules

As I talked to teams and customers, three clear personas emerged — not from a template, but from patterns in conversations:

1. **Business Analyst**
    - Wants to schedule recurring reports quickly.
    - Doesn't care how "clever" the system is, just wants it to work.
    - Overwhelmed by cryptic terminology and dense forms.
2. **IT Administrator**
    - Watches thousands of schedules, jobs, and failures.
    - Needs visibility and control, not surprises.
    - Uses workarounds and tools outside the product to track things.
3. **Power User**
    - Knows every quirk and workaround.
    - Can weave together complex distribution rules and dependencies.
    - Nervous that "modernization" will strip away control.

One thing was unexpectedly consistent:

Nobody defended the UI. Everybody defended the reliability.

That changed the design brief in my head. This wasn't *"Let's make it pretty."*
It was *"Let's modernize how it feels without breaking how it works."*

At the same time, I benchmarked Power Automate, Tableau Scheduler, Qlik NPrinting, and others. They offered sophisticated flows but often assumed a level of technical comfort many analysts didn't have.

That was our opportunity: **match the power, dial down the intimidation.**

The Real Problem: Not Just the UI, but the Journey

Once I mapped the existing journey, the problem came into focus.

To schedule a single report, a typical user might:

1. Right-click an asset in the Hub
2. Launch an external ReportCaster UI in a new tab
3. Navigate multiple forms and dialogues
4. Configure time, format, recipients, and distribution rules
5. Save, go somewhere else, and then hunt for that schedule later

It could easily take 10–15 clicks across 3 different tools. If you didn't already know where everything lived, it felt like playing UI treasure hunt.

From a UX standpoint, this translated to:

- Fragmented entry points
- Zero sense of "home" for scheduling
- Mental model mismatch between how users thought and how the system was structured

So I rewrote the problem for myself:

"How do we bring scheduling home to the Hub, reduce the cognitive burden, and still respect the complexity power users rely on?"

Setting the Strategy: Bring It Home to the Hub

I framed the strategy around four pillars:

1. **Embed scheduling in the Hub**
   No more jumping to an external legacy interface. The Hub is where users think, plan, and act — scheduling should live there too.
2. **Turn forms into guided flows**
   Old ReportCaster was a wall of fields. I replaced that with a step-based, guided journey showing only what mattered at each step.
3. **Design for both beginners and pros**
   New users should feel guided. Power users shouldn't lose reach or control.
4. **Anchor everything in a modern design system**
   This wasn't a facelift. It was a chance to align ReportCaster with the evolving WebFOCUS design system and accessibility standards.

This strategy became the lens we used to say *"yes"* or *"no"* to ideas — repeated in stakeholder reviews, engineer syncs, and design critiques.

Process: Unraveling the Old, Designing the New

1. Mapping the Forest

The first phase was painful and necessary.

- I walked through the old product screen by screen and noted what each control did.
- I traced how many clicks it took to complete common tasks.
- I captured edge cases — failure states, edit flows, distribution quirks.

It wasn't glamorous. It was forensic.

From that, we built a current-state workflow map — how a schedule was created, updated, monitored, and debugged.

Whenever someone said, *"It's not that bad,"* I could point to it and ask:

"Would you design it like this today?"

2. Designing the New Workflow

Next, I defined a streamlined **5-step flow** mirroring how people actually think:

1. Pick what you want to schedule (a report, dashboard, etc.)
2. Choose when and how often it runs
3. Decide who receives it and in what format
4. Confirm and save
5. Monitor and manage schedules from a central place

It meant collapsing multiple legacy controls, renaming things to match users' mental models, and negotiating with engineering around refactors.

3. From Sketches to Prototypes

I started with low-fidelity flows and wireframes:

- "What happens after they choose schedule type?"
- "Where do they see upcoming schedules?"
- "If something fails, what's the first thing they see?"

As alignment grew, I moved into high-fidelity prototypes that:

- Embedded scheduling UI directly in the Hub
- Used consistent design system patterns
- Introduced progressive disclosure — simple defaults for most users, advanced toggles for pros

We iterated dozens of times, trading off:

- When to show advanced filters vs. hide them
- Inline editing vs. separate detail panels
- How much data to reveal upfront

Instead of getting lost in the forest, I was carving a path for others.

Collaboration: Leading Without Owning the Org Chart

ReportCaster taught me a lot about **leading in spaces where UX wasn't the default center of gravity.**

Some realities:

- Long-time engineers were protective of a system that hadn't failed them.
- PM focus was split across initiatives — no clean UX ownership.
- Deadlines were set by release trains, not UX timelines.

So I leaned on what I do best — building relationships, listening, and being relentlessly clear.

I:

- Turned vague "we should modernize this" chats into concrete prototypes.
- Used support tickets and user pain as evidence.
- Invited engineers into early design reviews.
- Documented decisions so no one felt blindsided.

I wasn't just designing screens. I was designing alignment.

The Final Experience: Scheduling Belongs in the Hub

In the final experience:

- Scheduling starts directly in the Hub.

- The user sees a guided flow, not a maze of forms.
- Schedules can be viewed, edited, and managed in a unified dashboard.
- Errors and statuses are visible, not buried in logs.

Impact:

- 60% fewer clicks per scheduling task
- Scheduling time was cut roughly in half
- 3 separate tools merged into 1 cohesive flow
- 1-click access from Hub (down from 2+)

The sentiment changed from:

“I’m scared to touch that thing.”
To:
“This finally feels like part of the product, not an awkward bolt-on.”

What This Project Did to Me (In a Good Way)

On paper, this is a story about a legacy enterprise product.
In reality, it’s also a story about **me growing into a different kind of designer.**

This project taught me that:

- **Initiative matters.** Nobody asked me to own ReportCaster. I claimed it — and that opened doors to deeper trust, influence, and harder problems.
- **Leadership is a behavior, not a title.** I didn’t wait for perfect conditions. I led through clarity, persistence, and collaboration.
- **Enterprise UX is psychological.** Respect what long-time users and engineers love while still advocating for change.
- **The boring parts hold leverage.** Support calls, old specs, weird edge cases — those were the clues that made the design stronger.

If I had to summarize this project in one line:

I didn’t just redesign a scheduler. I redesigned how the team thought about UX in a 40-year-old product.

ReportCaster is now a foundation we can build on — and the trust, workflows, and patterns built here have fed directly into new initiatives like ML workflows and plugins.

UX isn’t a “nice to have.” It’s what makes all that complexity usable.