Designing for the Worst Day

MARCH 2026

14 min read • 3,157 words

Themes: Consciousness Technology Mental Health Programming Recursive Spiritual

You wake up and everything is loud. Not literally loud. Loud in the way that means your nervous system is running hot and every small decision feels like it has stakes. You need to pull data from a database for a report that's due in an hour. You need to send an email to someone important. You need to parse a datetime string into something a script can use. These are not hard tasks. You've done them a thousand times. But today your hands are shaking and your thoughts won't line up and the distance between knowing what to do and being able to do it feels like a canyon.

I've had this day. Many times. I have a serious mental health condition that I've written about extensively, and on my worst days, the gap between intention and execution is not a metaphor. It's a clinical reality. Bipolar disorder doesn't care that you know how to use an ORM. It doesn't care that you've been writing Python for fifteen years. When the episode hits, your executive function degrades, your working memory shrinks, and the tools you use every day either meet you in that diminished state or they don't.

Most of them don't.

This essay is about the ones that do, and why that matters for everyone, not just people with diagnoses.

The Principle

Design for the person having the worst day of their life.

Not the median user. Not the power user. Not the person reading your documentation on a calm Tuesday afternoon with a full cup of coffee and eight hours of sleep behind them. Design for the person whose hands are trembling and whose thoughts are scattered and who needs your tool to do one thing without making them feel stupid for not being able to figure it out.

If it works for that person, it works for everyone.

This is not a UX principle I borrowed from a design book. It came from being the person having the worst day. It came from reaching for a tool during a mental health crisis and discovering whether it was built with compassion or with assumptions about my capacity. The ones that assumed I was at my best failed me when I was at my worst. The ones that didn't assume anything about my state just worked.

> This principle appears on my worldview page as "Design for your worst day." It's not a design heuristic. It's a survival lesson that turned out to generalize beautifully to software.

I've been building software this way for over a decade. Requests, Tablib, Records, Maya. Every one of them was shaped by the same question: what would this feel like if I couldn't think straight? The answer to that question produces better software than any amount of user research conducted on people having good days.

What Bad Days Actually Look Like

I need to be specific here, because "designing for accessibility" has become a vague corporate virtue that rarely gets translated into concrete practice.

On a bad day with bipolar disorder, here's what degrades. Working memory shrinks. You can hold fewer things in your head at once. Abstract reasoning gets harder. You can understand concrete steps but struggle to navigate layers of indirection. Decision fatigue hits faster. Every fork in the road costs more energy than it should. Error recovery is expensive. When something goes wrong, the cognitive cost of figuring out what happened and how to fix it can be enough to make you stop entirely.

This is not unique to bipolar disorder. Depression does this. Anxiety does this. ADHD does this. Grief does this. Sleep deprivation does this. A bad fight with your partner does this. Being a new parent does this. Being human, on certain days, does this.

When I say design for the worst day, I mean design for the universal human experience of diminished capacity. Everyone has these days. People with psychiatric conditions just have them more often and more severely, which makes us excellent testers for whether your tool actually works when it matters.

Tablib: The Data Knows How to Represent Itself

Tablib was one of my first open source projects. The idea was simple: a dataset should know how to become things. You shouldn't have to manage the translation between formats. You put data in, you get data out, in whatever shape you need.

```
import tablib

# Create a dataset. This is the hard part. (It's not hard.)
data = tablib.Dataset(headers=["name", "age", "city"])
data.append(["Alice", 24, "New York"])
data.append(["Bob", 30, "San Francisco"])

# Now the data knows how to be anything.
print(data.csv)
print(data.json)
print(data.yaml)

# Need an Excel file? The data knows.
with open("people.xlsx", "wb") as f:
    f.write(data.xlsx)
```

Compare this to what you'd normally do:

```
import csv
import json
import io

# Without Tablib: you manage everything.
# You are the translator. You hold the context.
# You remember the format. You handle the edges.

data = [
    {"name": "Alice", "age": 24, "city": "New York"},
    {"name": "Bob", "age": 30, "city": "San Francisco"},
]

# Want CSV? Build a writer. Manage a buffer. Write headers.
# Remember that DictWriter needs fieldnames explicitly.
output = io.StringIO()
writer = csv.DictWriter(output, fieldnames=["name", "age", "city"])
writer.writeheader()
writer.writerows(data)
csv_string = output.getvalue()

# Want JSON? Different API. Different mental model.
json_string = json.dumps(data)

# Want Excel? Install openpyxl. Learn a new API.
# Create a workbook. Get the active sheet.
# Write headers manually. Iterate rows.
# Remember that Excel is 1-indexed. (Why?)
```

On a good day, the second approach is fine. Tedious, but manageable. On a bad day, every one of those steps is a place to lose your thread. Every format-specific API is a new context your working memory has to hold. Every edge case is a trap.

Tablib removes the translation burden. The data knows how to represent itself. You don't have to manage that. When your brain can't hold complexity, Tablib doesn't ask you to.

> This is what I mean by "format agnostic" in Tablib's design. It's not a technical feature. It's a compassionate one. The data adapts to what you need instead of forcing you to adapt to what it requires.

Records: Just Write the Query

Records exists because ORMs are abstraction layers, and every abstraction layer is a filter that eliminates people having bad days.

When your brain is working well, an ORM is lovely. You define models, build querysets, chain filters, and the ORM translates it all into SQL for you. Beautiful. But when your brain is not working well, those layers of indirection become walls. You know the SQL you want to write. You can see it in your head. But you can't remember the ORM syntax to express it, and the gap between what you know and what the tool demands is where you give up.

Records strips away everything between you and your data.

```python
import records

db = records.Database("sqlite:///my.db")

# You know SQL. Just write SQL.
rows = db.query("SELECT * FROM users WHERE active = 1")

# Iterate. That's it.
for row in rows:
    print(row.name, row.email)

# Need it as CSV? Tablib integration, built in.
print(rows.export("csv"))
```

No model definitions. No migration files. No queryset chains. No manager objects. You connect, you query, you get results. When you can't hold abstraction layers in your head, Records doesn't make you.

The interface meets you where you are, not where it wishes you were.

That sentence is the entire philosophy. Records doesn't wish you were the kind of developer who has an elegant ORM schema defined. It assumes you're the kind of developer who knows what data you want and wants to get it with the least possible friction. On your worst day, that's exactly the kind of developer you are.

> I want to be clear: ORMs are good software. SQLAlchemy is brilliant. Records is built on top of it. The point isn't that abstraction is bad. The point is that abstraction should be available, never required. Power users can reach for the ORM. Everyone else can just write the query.

Maya: Say What You Mean

Datetime APIs are famously hostile. Python's built-in `datetime` module is powerful and comprehensive and absolutely punishing on a bad day. Timezones, naive vs. aware objects, `strftime` format codes, the difference between `datetime.datetime.now()` and `datetime.datetime.utcnow()` and why one of them is almost always wrong.

Maya lets you speak like a human.

```
import maya

# What time is it?
now = maya.now()

# When is tomorrow?
tomorrow = maya.when("tomorrow")

# When is next Friday?
friday = maya.when("next friday")

# Parse something a human wrote.
meeting = maya.when("March 20th at 3pm")

# Get a nice human-readable string.
print(now.slang_time())  # "just now"
```

Compare:

```
from datetime import datetime, timedelta, timezone

# What time is it? (But which "now"? Naive or aware?
# Local or UTC? This matters and it's not obvious.)
now = datetime.now(timezone.utc)

# When is tomorrow?
tomorrow = now + timedelta(days=1)

# When is next Friday? Good luck.
# You need to calculate the day offset.
today = now.weekday()
days_ahead = 4 - today  # Friday is 4
if days_ahead <= 0:
    days_ahead += 7
next_friday = now + timedelta(days=days_ahead)

# Parse something a human wrote.
# Hope you remember the format codes.
meeting = datetime.strptime("March 20th at 3pm", "%B %dth at %I%p")
# (This won't actually work. Good luck with "th".)
```

When you can't think in ISO 8601 timestamps, Maya doesn't make you. When your brain can't compute weekday offsets, Maya doesn't ask you to. You say what you mean in language that sounds like thinking, and Maya figures out the rest.

Simple defaults aren't dumbing down. They're compassion compiled.

LLMs: The Frontier of Worst-Day Design

Here's where the principle extends beyond libraries I've built and into the most significant shift in computing I've seen in my career.

Large language models are the best worst-day technology that has ever existed.

When I'm in a bad mental state, there are days when I can't draft an email. Not because I don't know what I want to say, but because the executive function required to organize thoughts into grammatically correct sentences and arrange those sentences into a coherent message is simply not available. The gap between intention and expression becomes impassable.

> Executive function is the cognitive process that handles planning, organizing, initiating, and completing tasks. It's one of the first things to degrade during mood episodes, sleep deprivation, or extreme stress. It's also invisible. You look fine. You just can't do the thing.

On those days, I can throw tokens at an LLM that aren't even grammatically sane. Fragments. Quasi-words. Half-formed intentions smeared across a text box. And it understands. Not approximately. Exactly. It takes the mess of my diminished output and reconstructs the meaning I was trying to express. It drafts the email I couldn't draft. It becomes a dependency API for executive function.

```
# What I can produce on a bad day:
input = """
    email to james re: meeting
    cant do thursday
    maybe next week?? the project thing
    need to also mention budget idk
"""

# What an LLM produces from that:
output = """
    Hi James,

    Thanks for the meeting invite. Unfortunately Thursday
    doesn't work for me,would sometime next week be
    possible instead?

    I also wanted to flag that we should discuss the
    project budget when we do connect. Happy to bring
    some numbers.

    Best,
    Kenneth
"""

# The interface met me where I was.
# Not where it wished I were.
```

This is worst-day design at its most profound. The interface doesn't require coherence. It doesn't require grammar. It doesn't require executive function. It takes whatever you can produce and meets you there. It doesn't judge the gap between your best self and your current self. It just bridges it.

I've written about AI as accessibility device before, but I want to be more direct here. For people with mental health conditions, with ADHD, with autism, with any neurodivergent wiring that creates a gap between knowing and doing, LLMs are not a luxury. They are a prosthetic for executive function. They are the ramp to the building that used to only have stairs.

And they work this way because, whether the designers intended it or not, they embody the worst-day principle. The model doesn't assume you're at your best. It works with whatever you give it. That's the design. That's why it matters.

The Principles

I want to state these clearly, because they're the output of fifteen years of building software while managing a condition that regularly puts me on the user side of bad design. These aren't theory. They're field notes.

The interface should meet you where you are, not where it wishes you were. If someone has to be at their cognitive best to use your tool, you've designed for a fantasy user who doesn't exist most of the time.

If it requires your best self to use, it's not designed for humans. Humans are not at their best most of the time. We're tired. We're distracted. We're worried about something that has nothing to do with your software. Design for that person. They're the real user.

Complexity should be available, never required. Power users can reach for the advanced features. Everyone else should be able to do the basic thing without climbing through abstraction layers to get there.

Simple defaults aren't dumbing down. They're compassion compiled. Every time you make the simple path easy, you're making a statement about who deserves to use your tool. Make that statement generous.

The best tools disappear. You don't think about the tool. You think about the work. If someone is thinking about your API instead of their problem, your API has inserted itself between a person and their goal. Get out of the way.

Error messages should help, not blame. "Invalid input" is an accusation. "Did you mean..." is a hand extended. The difference between those two responses is the difference between a tool that shames you on your worst day and one that catches you.

> I think about Python's evolution here. Python 3.12 introduced error messages like "Did you mean 'print'?" instead of just "NameError." That single change turned a dead end into a guidepost. Someone on the Python core team understood this principle, even if they wouldn't have framed it this way.

Every layer of abstraction you force someone through is a filter that eliminates people having bad days. Not by malice. By friction. Each layer costs cognitive energy, and people having bad days have less of it. If you require five steps where two would do, you've decided that three steps worth of people don't matter.

Design for trembling hands and scattered thoughts, and steady hands will thank you too. This is the universality of the principle. Curb cuts were designed for wheelchairs and everyone uses them. Worst-day design works the same way. What serves the struggling serves everyone.

Why This Goes Beyond UX

I could frame all of this as user experience advice and it would be useful. But that framing would miss the deeper point.

When you design for the worst day, you're making a statement about who matters. You're saying: the person who is struggling right now, whose brain is betraying them, whose hands won't stop shaking, whose thoughts won't line up into sentences, that person deserves to use this tool. That person deserves to get their work done. That person deserves to participate.

Most software makes the opposite statement, quietly, through accumulated friction. It says: this tool is for people who are functioning well. If you're not functioning well, come back when you are. The cruelty of that statement is that it's never made explicitly. It's embedded in the interface. In the five-step configuration. In the error message that says "invalid" without saying what would be valid. In the abstraction layer that assumes you can hold the whole model in your head.

I've written about the recursive loop between code and consciousness. The idea that programmers shape code, code shapes minds, and therefore programmers shape collective consciousness. Worst-day design is where that loop meets its most consequential test. Because the tools we build don't just shape how people think on good days. They shape whether people can think at all on bad days. They determine who gets to participate and who gets filtered out.

The algorithm eats virtue when systems optimize for engagement over wellbeing. But exclusion is subtler than exploitation. You don't have to optimize against someone to exclude them. You just have to not think about them. You just have to design for the median case and let the edges fall away.

The worst day isn't an edge case. It's a Tuesday. It's a new parent who hasn't slept. It's a person with depression who used all their energy getting out of bed. It's a developer with ADHD whose medication isn't working today. It's you, on the day you'd rather not talk about. These people aren't outliers. They're your users, having a human experience.

Design for them. They'll remember.

The Practice

Here's how I actually do this, concretely, when I'm building something.

I write the API call I'd want to make on my worst day. Before I write any implementation, before I think about architecture or edge cases or performance, I write the code I'd want to type when my hands are shaking and my brain is fog. That becomes the interface. Everything else is implementation detail.

```
# Step 1: Write the call you'd want to make
# on your worst day.

import records
db = records.Database()
rows = db.query("SELECT * FROM users")
print(rows.export("csv"))

# Step 2: Make that work.
# Whatever it takes behind the scenes,
# make exactly that work.

# Step 3: Add complexity for people who want it.
# But never require it.
# The first version, the worst-day version,
# should always keep working.
```

This is programming as spiritual practice in its most concrete form. Not abstract meditation on code. Actual compassion, compiled into an interface, serving the person who needs it most.

I told you at the beginning that this principle isn't influenced by my mental health. It IS my mental health turned into a design philosophy. Every panic attack that made a complex API unusable. Every depressive episode that reduced my cognitive bandwidth to a narrow beam. Every manic crash that left me staring at a screen, knowing what I needed to do but unable to navigate the steps to do it. Those experiences are the research. The libraries are the findings.

Design for the person having the worst day of their life. If it works for them, it works for everyone.

That's the whole philosophy. Everything else is implementation.

This essay is the prescriptive companion to Open Source Gave Me Everything Until I Had Nothing Left to Give and The Lego Bricks Era. For the philosophical foundation, see From HTTP to Consciousness and The Recursive Loop. For the libraries referenced, see Tablib, Records, and Maya. For the worldview that underlies all of it, see Worldview.